ACE-GCN: A Fast Data-driven FPGA Accelerator for GCN Embedding

JOSÉ ROMERO HUNG, CHAO LI, PENGYU WANG, CHUANMING SHAO, JINYANG GUO, JING WANG, and GUOYONG SHI, Shanghai Jiao Tong University

ACE-GCN is a fast and resource/energy-efficient FPGA accelerator for graph convolutional embedding under data-driven and in-place processing conditions. Our accelerator exploits the inherent power law distribution and high sparsity commonly exhibited by real-world graphs datasets. Contrary to other hardware implementations of GCN, on which traditional optimization techniques are employed to bypass the problem of dataset sparsity, our architecture is designed to take advantage of this very same situation. We propose and implement an innovative acceleration approach supported by our "implicit-processing-by-association" concept, in conjunction with a dataset-customized convolutional operator. The computational relief and consequential acceleration effect arise from the possibility of replacing rather complex convolutional operations for a faster embedding result estimation. Based on a computationally inexpensive and super-expedited similarity calculation, our accelerator is able to decide from the automatic embedding estimation or the unavoidable direct convolution operation. Evaluations demonstrate that our approach presents excellent applicability and competitive acceleration value. Depending on the dataset and efficiency level at the target, between 23× and 4,930× PyG baseline, coming close to AWB-GCN by 46% to 81% on smaller datasets and noticeable surpassing AWB-GCN for larger datasets and with controllable accuracy loss levels. We further demonstrate the unique hardware optimization characteristics of our approach and discuss its multi-processing potentiality.

CCS Concepts: • **Hardware → Hardware accelerators**; • **Computer systems organization → Neural networks**; **Real-time system architecture**; • **Mathematics of computing → Approximation algorithms;**

Additional Key Words and Phrases: Graph convolutional neural networks, GCN, graph processing, graph embedding, node identification, FPGA, embedded systems, data-driven, sparse datasets, power law distribution

ACM Reference format:
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, Article 21 (September 2021), 23 pages.
https://doi.org/10.1145/3470536

1 INTRODUCTION

Graph Neural Networks (GNNs) designate all the research efforts to capitalize on the predicting and classifying power of **deep learning neural networks (DLNNs)** for graph-structured data

This work is supported by the National Key Research & Development Program of China (Grant No. 2018YFB1003503).
Authors' addresses: J. Romero Hung, C. Li (corresponding author), P. Wang, C. Shao, J. Guo, J. Wang, G. Shi, Shanghai Jiao Tong University, 800 Dongchuan Road, Shanghai, China, 200240; emails: romerohung@sjtu.edu.cn, lichao@cs.sjtu.edu.cn, wpybtwsjtu.edu.cn, cyunmingsjtu.edu.cn, lazarussjtu.edu.cn, jing618sjtu.edu.cn, shiguoyong@sjtu.edu.cn.
Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2021 Association for Computing Machinery.
1936-7406/2021/09-ART21 $15.00
https://doi.org/10.1145/3470536

mining [19, 20]. Recently, the field has received much attention due to its demonstrated potentialities in diverse practical scenarios [9, 15, 36, 39]. However, due to the complexity of processing multiple and independent relational information along with its propagative effect, traditional CPU/GPU-software implementation still makes any in-place GNN deployment unpractical/unfeasible.

GCN is one of the most popular variances of GNN. The current state of the art on GCN algorithmic research keeps reporting extremely long processing times and speed/scalability issues for non-Euclidian datasets [42]. To make things worse, achieving **convolutional neural network (CNN)** hardware acceleration still represents a challenge on its own [21, 33], and because of the differences between Euclidean CNN and non-Euclidean GCN source data, traditional pruning and quantization methods are hardly applicable. The result is a general lack of original hardware acceleration proposals for GCN, the few of them focusing on coping with the issues arising from blindly adapting rather inflexible architectures into graph structures, consequently inheriting implementation challenges from both families.

However, the same differences that preempt GCN from directly recoursing traditional CNN acceleration methods [18] could grant unexplored advantages on the GCN side. It is possible to embrace and exploit the specific nature of real-world graphs into more adequate acceleration structures. Most real-world graph datasets exhibit strong power law distribution patterns with high sparsity levels. The number of structural sub-graphs matching events becomes more frequent as the graph grows. Then, the definitive similarity estimation is susceptible to deterministic variables and design parameters according to targeted specifications and resource availability.

Embedding and aggregation are, by definition, the most important and iterative processes inside GCN; their acceleration is a common concern and basepoint of many state-of-the-art architectures out there [5, 8]. In this article, we propose ACE-GCN: a fast data-driven FPGA **A**ccelerator for graph **C**onvolutional neural network **E**mbedding based on graph structural similarity.

Our similarity core derives from a vector-based Jaccard graph similarity coefficient. With it, ACE-GCN is able to recognize similar graph structures and automatically produce faster and inexpensive embedding estimations. This ultimately promotes a reduction over the neural network workload and accelerates inference completion, albeit with some controlled accuracy loss. This article makes the following key contributions:

(1) Analyze the theoretical connection between the structural characteristics of real-world graphs, their feature-based similarity coefficients, and their significance for graph convolutional propagation
(2) Implement an innovative GCN embedding hardware accelerator based on detection and recoursing of sub-graph types by structural similarity estimation, with data-driven characteristics and modifiable performance according to resource availability
(3) The quantitative and qualitative experimentation that demonstrate the virtues and possible issues of our approach as a data-driven resource conservative accelerator, and the effect of variance of storage capacity as a graph convolution operation reliever

The rest of this article is organized as follows: In Section 2, we develop the background, previous work, and motivations of our study. In Section 3, we introduce the theoretical principles that rule our design. In Section 4, ACE-GCN architecture and dataflow are described. Section 5 details the experimental setup. Section 6 is the performance analysis. Finally, in Section 7, we present our conclusions.

2 BACKGROUND AND MOTIVATION

Unlike contextual processing where data is required to be provided as a whole before operations start, in data-driven processing, a constant, mutable, and uneven data provision may broadcast

from a complex set of independent sensorial devices [4]. Its receiving component must evolve as new data arrives. Thus, it cannot depend on a centralized or stochastically expected memory allocation. This way, data-driven implementations are generally associated with some level of in-place processing benefits [13].

Early works already acknowledged interception-based similarity coefficients as a way to minimize communication and computing overload, for example, [2, 31]. The last one proposed a data-driven approach for mining physiological data on which a similarity coefficient automatically allows clustering among sets of rules. The adopted similarity function calculates an overlapping ratio. The same is defined as $Over(R_1, R_2) = |R_1 \cap R_2|/|R_1 \cup R_2|$, where R_1 and R_2 are vectors representing a collected set of rules.

Data-driven hardware implementation has a natural correspondence with FPGA. This device is primarily a tool from the signal processing world. It contains simpler communication protocols than GPU [26] and, as a consequence, faster signal response/lower latency. Unlike ASICs, FPGA is highly customizable and modular, able to match with the most different computing environments.

The continuous research on the potentialities of DNN into the enhancement of fundamental applications has derived into the emerging field of GNNs, including Weisfeiler-Lehman Graph Kernels [29], Graph Attention Networks [32], and Graph Convolutional Neural Network or GCN [16], among others. In essence, GNN proposes specific DL configurations to learn the implicit behavior of complex graphs. This results in better interpretable results on inference tasks such as node classification and node future link prediction [41].

Due to its demonstrated applicability and flexibility, the particular approach of GCN [16] has deserved notoriety in the last years. Consider an undirected graph represented by the expression $G = (V, E)$ with n total number of nodes, V set of vertices, and E set of edges. By declaring a set of feature matrices like $X = [X_1, X_2, \ldots, X_u]^T \in R^{n \times u}$ with a u-dimensional feature vector, the resulting neural propagation rule aggregates neighboring graph data with the equation $X^{(l+1)} = \sigma(D'^{-1/2}A'D'^{-1/2}X^{(l)}W^{(l)})$, where $X^{(l)}$ is the feature matrix of the current convolutional layer, $W^{(l)}$ is the matrix of trainable weights, $W^{(l+1)}$ is the expected feature matrix output (and input to the next convolutional layer), $\sigma(.)$ is the activation function, and $D'^{-1/2}A'D'^{-1/2}$, or just A, is the normalized adjacency matrix. The output is a new map of features $n \times u$ with reduced dimensional representation.

One of the most iterating processes of GCN is graph embedding. This consists of encoding a graph and its associated node features into a lower-dimensional representation. The process is carried throughout several convolutional layers until the information is fit enough for automatic classification [16]. The **learnable graph convolutional network (LGCN)** [6] included a novel method denominated **"k-largest node selection" (KLNS)**. This heuristic-model mixed technique greatly simplifies graph learning by sorting out the largest features of a k-limited sub-graph representation, promoting the transformation of generic graphs into a more processable grid-like type of structure.

More specifically, A and $X^{(l)}$ are morphed into a new $\widetilde{X}^{(l)} \in \mathbb{R}^{(k+1)\times u \times c}$ feature space representation through the function $\widetilde{X}^l = g(k, A, X^{(l)})$ with k as abstraction hyperparameter, u size of feature space, and c number of channels. The embedding locality information updates feature representation for the next layer s.t. $X^{(l+1)} \in \mathbb{R}^{k \times u}$, in practice, reducing the spatial size from $k + 1$ to 1, and then the lineal convolution is:

$$X^{(l+1)} = X^{(l)}W^{(l)}. \tag{1}$$

It is widely known [17, 22, 24, 25, 27], that when real-world graphs tend to have larger sizes, they acquire power law distribution and high levels of sparsity. Studies like [10, 38] have further explored this fact, identifying similarity ratios among sub-graphs by exploiting vertex exchangeability characteristics. The properties of high sparsity in large graphs can be expressed from the

relation that describes its vertex degree probability, i.e., $P(k) = Ck^{-\alpha}$. Thus, for any scale-free network, the probability of a random vertex connecting to k-number of neighbors is C proportional to $k^{-\alpha}$ for some fixed constant α.

Although graphs lack the explicit locality of pixel-based datasets, in highly sparse graphs each node is inclined to connect to a limited number of nodes and not to all the nodes in the graph. This situation becomes more evident as the graph becomes larger. In the vast majority of graph processing acceleration papers [14, 23, 40], this effect has already been recognized as a challenge on performance. The common solution to this involves exploiting customized architectures to achieve complex transversal stochastic schedules. This type of approach barely manages to amend the poor locality that greatly affects multi-processing balance.

The same concerns to obtaining architecture regularity on the irregularity provoked by the sparsity of data have been inherited by the already few proper GCN hardware accelerators out there; worthy of mention are [12], HyGCN [37], and AWB-GCN [8]. These studies start by assuming source data under precise stochastically expected locations. However, such ideal conditions on controlled computing environments do not match more realistic data-driven setups like [34]. A different data-driven perspective for graph inference acceleration becomes necessary.

Interestingly, GCN accelerators have so far not explored principles of graph similarity that could otherwise provide useful discriminative information. Similarity, as an acceleration mechanism, is regularly known to data-driven solutions [2]. Considering two sub-graphs $Z(a)$ and $Z(b)$, a measure of similarity between both can be calculated using a vector-based Jaccard similarity coefficient. This coefficient quantifies the ratio between interception and union of the set of features corresponding to both sub-graphs, as described:

$$J(a,b) = \frac{|Z(a) \cap Z(b)|}{|Z(a) \cup Z(b)|}. \tag{2}$$

If both sub-graphs are described in terms of feature vectors like $a = [a_1, a_2, \ldots, a_u]$ and $b = [b_1, b_2, \ldots, b_u]$, the Jaccard similarity coefficient can be calculated by dividing the summation of all minimum features $\min(a_i, b_i)$ between all maximum features $\max(a_i, b_i)$ through

$$J(a,b) = \frac{\sum_{i=1}^{u} min(a_i, b_i)}{\sum_{i=1}^{u} max(a_i, b_i)}. \tag{3}$$

In the literature, we find the graph similarity relation with CNN presented in three types: first, CNN applied to assist in the calculation of graph similarity, totally replacing any stochastic method [1]; second, stochastic methods like graph intersection and BFS assisting CNN in prediction tasks [9, 30]; and finally, stochastic methods totally replacing CNN for prediction and classification tasks [20, 28].

Like in the multi-media skeleton identification of [9], fast-streamed small-sized data applications have a preference for the second option, while massive contextual processing generally opts for the first. It is possible to leverage some of these concepts on similarity [20] in an effort to accelerate GCN inference operations.

3 ACE-GCN DESIGN PRINCIPLES

Consider an isolated sector of a highly sparse graph composed of some nearing one-hop sub-graphs as shown in Figure 1 (left). According to their degree and set of features, populating nodes may be implicitly related by a convergence coefficient named "τ" (as "type"). If their associated features are considered equivalents, then the resulting embedding inherits the same equivalence. Moreover, if they differ by a ratio, this value can be capitalized as an accuracy marker.

Positive convergence events can be considered as centroids, cataloged as "types," and stored for future utilization. The probability of matching such types should be highly representative on

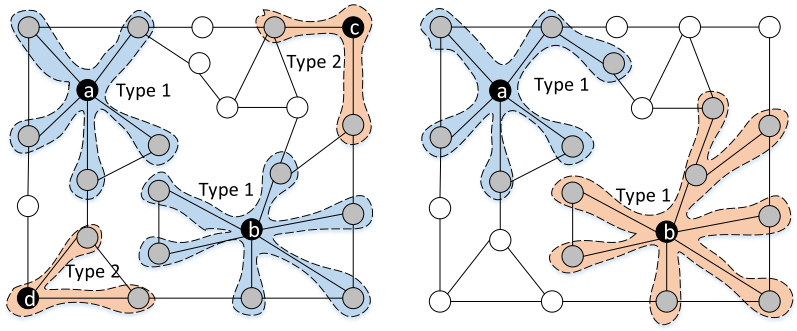

Fig. 1. Graphic representation of the principle of sub-graph similarity for implicit-processing-by-association of datasets with power law distribution. **(Left)** A one-hop node association; each sub-graph-in-observance or SIO is a candidate to form up a new centroid or "type"; types are represented here in different colors. In our work, each SIO is composed of the reference node (RN) and at least k neighboring nodes (NNs). For less than k neighboring nodes, the remaining positions are padded with pre-calculated, lower-bounded, average dataset values. Depending on the similarity of their features, other SIOs can be related to one of the detected types and its associated embedding results. **(Right)** The same approach can be applied to successive n-hop upper-level embeddings.

real-world graphs. If a node connects to a number of neighbors with a determined set of features, and then another unrelated node connects to the same amount with "similar" features to the first case, then both nodes are considered to have a degree of similarity. In this case, any of the nodes can become a type and its ratio of positive matching stored. The same principle could be extended to successive hops besides one-hop as shown in Figure 1 (right).

A graph sequence is considered sparse if $|E_n| = o(|V_n|^2)$, where the number of edges is asymptotically upper bounded by $C|V_n|^2$ for all constants C. For any sparse sub-graph, its similarity number of events is proportional to the graph size and can be bounded by a similarity window. Let us assume $Sim(a, b)$ denotes a similarity absolute detector between a pair of sub-graphs $Z(a)$ and $Z(b)$, induced by their respective Jaccard similarity coefficient s.t.

$$Sim(a,b) = \begin{cases} 0 & \text{if } J(a,b) \leq P_1 \\ 1 & \text{otherwise,} \end{cases} \tag{4}$$

where P_1 is a similarity threshold, in practice filtering $J(a, b)$. The number of similarity events detected for a sub-graph a, regarding a second sub-graph b, is proportional to the graph size through a constant τ so that whenever b tends to ∞, the number of similarity events tends to ∞/τ, in essence, a linear correlation. From this, it can be concluded that, having a specific sub-graph and P_1 threshold, the larger the graph, the larger the tendency for positive sub-graph similarity detection, formally described with the relation

$$\sum_{n=0}^{\infty} Sim(a,b) \rightarrow \frac{\infty}{\tau}. \tag{5}$$

Since similarity is here determined by an interval, sub-graph equivalence has an inherent accuracy loss named Q, and at this point, any similarity coefficient is going to be driven by the shape of data, in this case, power law distribution of $J(a, b)$. Thus, a similarity error can be associated by

$$Q \rightarrow C * \frac{1}{J(a,b)^{\tau}}, \tag{6}$$

Table 1. List of Acronyms Utilized in This Work and Their Meaning

Acronym	Meaning	Acronym	Meaning
SIO	Sub-graph-in-observance	DSIOFV	Detected SIO feature vectors
TIO	Type-in-observance	SIORNFV	SIO reference node feature vector
RN	Reference node	CLR	Classification results
NN	Neighboring node	GATH	Gatherer
GTL	Graph tracking list	RL	Reduction layer
TSIO	Tiled sub-graph-in-observance	CL	Classification layer
CFV	Classification feature vector	BN	Batch normalization
ID	Node identification number	TC	Type creator
FC1W/FC2W	Fully connected layer weights	TM	Type matcher
FGAM	Full graph adjacency matrix	HSE	High-similarity estimator
FGFM	Full graph feature matrix	BT	Bank of types
PVE	Prevalence estimator	BE	Bank of embedding
PVL	Prevalence estimator list	KLNS	k-largest node selector

on which the accuracy loss Q has a tendency to become negatively proportional to the similarity coefficient $J(a, b)$ at a fixed constant τ multiplied by a proportionality constant C.

The property of vertex exchangeability is defined as the invariance of the distribution of any finite sub-matrix corresponding to any finite collection of vertices under finite permutation [3]. Consider a random sequence of sub-graphs Z_n with random sequence of edges described by $V_n = \{1, \ldots, n\}$; let π be any permutation of the integers n. Z_n is infinitely vertex exchangeable if for every $n \in \mathbb{N}$ and every π permutation of the vertices n, Z_n is approximately $\widetilde{Z}_n$, where $\widetilde{Z}_n$ has n vertices and π edges.

Then, if both sub-graphs have a range of similarity, vertex feature characteristics are mutually exchangeable. Lineal convolution operations produced by such set of features can be considered equivalent and approximative.

If $Z(a)$ has a level of equivalence to $Z(b)$, then the convolution of their components A and X are proportional to an error Q. With a common linear activation kernel as $W_a^{(l)} = W_b^{(l)}$, convolution equivalence can be represented by a relation like $A_a X_a^{(l)} * Q = A_b X_b^{(l)}$, with the definitive propagation form limited by

$$X_a^{(l+1)} * Q = X_b^{(l+1)}. \tag{7}$$

Considering Equation (6), it is possible to find levels of structural similarity among sub-graphs in enough quantities to justify the storing of matching samples in the form of centroids, especially as the graph becomes larger. Such levels of similarities can also be seen as a dissimilarity range, associated with an error coefficient that renders into a potential accuracy loss, and then its value can be shaped through a set of parameters (Equation (7)).

From the exchangeability property of graphs, two sub-graphs can share the same convolution embedding (Equation (8)), which is proportional to their mutual level of similarity. Because of the sparsity, it is expected that the stored centroids become highly representative, meaning that a substantially smaller number of types contribute to infer a much larger occurrence of embedding operations.

The process of detecting and optimizing such centroids bears analogies with the feature learning process from machine learning; we also refer to it as centroid learning. For the ease of the reader, Table 1 contains the most reiterative acronyms utilized throughout this article.

Due to its particular ability of creating highly representative sub-graphs, we select LGCN as our basis algorithm variance. Its simplified approach to the neural stages may allow us to properly isolate the neural processing resources from the heuristic-based elements. The possibility of a gradual

Table 2. Memory Sectors Breakdown with Parametrized Shape and Size in Mb. Sram On-chip (Light Blue), sdram Off-chip (Light Green), and Sdram Source Off-chip (Dark Green)

Sector	Shape	Cora	CiteSeer	PubMed	NELL //...// Reddit
FGFM	N^*u^0				
FGAM	N^*N	31.9611 MB	99.9426 MB	127.4630 MB	3388.4455 MB
FC1W	N^*u^1				
BT	$(k+1)^*u^{1*}\tau_3$				
DSIOFV	N^*u^1				
BE	$u^{8*}\tau_3$				
CLR	N*u8	78.8706 MB	78.9334 MB	86.8522 MB	298.3281 MB
FDET 1A	$((k/2)+1)^*u^{1*}\ u^2$				
FDET 1B	$((k/2)+1)^*u^{2*}u^3$				
FDET 2A	$((k/2)+1)^*u^{4*}u^5$				
FDET 2B	$((k/2)+1)^*u^{5*}u^6$				
TSIO	$(k+1)^*u^1$				
DSIOFVΔ	$N^{b*}u^0$				
SIORNFV	1^*u^0				
GTL	$N^*(P_3^b+P_4^b)$				
BT^Δ	$(k+1)^*u^{1*}\tau^\Delta$				
BE^Δ	$u^{8*}\tau^\Delta$				
CFV	$C^{b*}u^{0*}P_2$				
$FC1W^\Delta$	$(k+1)^*u^0$				
FC2W	$u^{7*}u^8$				
PVL	$\tau_3\ ^*(N^b + P_3^b + P_4^b)$	1.2463 MB	2.1256 MB	2.7618 MB	15.3518 MB

Four different ACE-GCN configura-tions are generated: Three customized for the smaller datasets Cora, CiteSeer, and PubMed and a big one forthe rest, from NELL to Reddit. Each configuration is designed with different parameter values, according toperformance goals.

relief of neural computing power implies that such unused capacity could be further exploited by more traditional parallel configurations. This way, we take advantage of LGCN characteristics to introduce new elements dedicated to our own proposed "implicit-processing-by-association" approach.

4 ACE-GCN ARCHITECTURE

4.1 Main Components Description

ACE-GCN is a composition of a set of functionally independent modules. Every module is controlled by levels of local **finite state machines (FSMs)**, which in turn are centralized into a top FSM. The top FSM serves as a module scheduler and as memory traffic manager for optimal data synchronization within the IP. A breakdown of the memory sectors and their sizes is shown in Table 2. A description of the most important modules within the architecture is as follows:

- ***GATH:*** The purpose of the gatherer or GATH is to collect, detect, and efficiently process a continued random provision of graph information. GATH will launch a node inference only when enough information related to that node has arrived at the system (an SIO has been detected), this without halting the entire process and while GATH keeps receiving and collecting other nodes' information.
- ***KLNS:*** The k-largest node selector is based on the homonym model in Equation (1) proposed by [6]. A representation of the coordination between the internal **graph tracking**

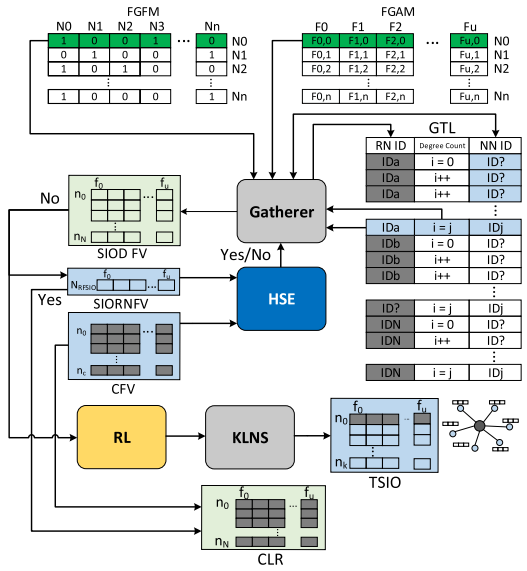

Fig. 2. The objective of the SIO gathering and tiling process is partitioning a full raw graph into smaller sub-graphs with properly representative feature vectors and within realistic memory broadcasting conditions. The graph information is sent from the source full graph feature adjacency matrix (FGFM and FGAM) one node at a time. The gatherer (GATH) receives the first node feature vector according to the streamed binary adjacency vector composition. GATH is supported by a tracking list named GTL at SRAM. GTL constantly updates the degree information for each node. It keeps a record of each arriving ID plus their 1-hop connected IDs and degree achieved; such connection list is obtained following the adjacency vector that follows the node feature vector. The node information is transferred back to the DSIOFV sector through its DSIOFV$^{\Delta}$ prefetching buffer. Whenever an SIO has collected enough information (at least j NNs have been received), it triggers a primordial similarity estimation from HSE (before dimensional reduction takes place). Based on this, it can automatically produce the feature vector estimation (without passing through any neural layer) or start the longer path to graph embedding. For this, the detected SIO is processed by RL, and then KLNS ranks the k-largest features from the set of neighbors. The operation is done directly over TSIO sector resources. For the parameter k, we stick to the best performance report in [6] with k = 8 for all datasets of similar sparsity degree.

list (GTL) and KLNS is further explained in Figure 2. This module produces the grid-shape transformation of SIO that we refer to as "tiled" SIO, stored at TSIO memory sector, which is indispensable for further processing.

- ***HSE:*** The **high-similarity estimator (HSE)** is the core module within the similarity estimation circuit. It performs a fast comparison between a pair of tiled sub-graphs, i.e., between the TSIO produced by KLNS and the currently parading **type in observance (TIO)**. The maximum number of types to be compared is denominated "τ" and is a design parameter on storage capacity granted to the accelerator.
  The objective is to output a definitive signal that acknowledges whether the pair is similar or not based on their constituent vectors. Internally, this comparison produces a vector-based Jaccard similarity coefficient that is filtered by a threshold parameter P_1 as in Equations (4) and (5). Figure 3 further details the architecture of HSE and the centroids parading system, from the arrival of the feature vectors to the similarity decision signal (1 or 0).
  The types parading memory system that feeds HSE follows a time-division-multiplexed scheduling. This allows us to perform continued partial reconfiguration of SRAM sectors

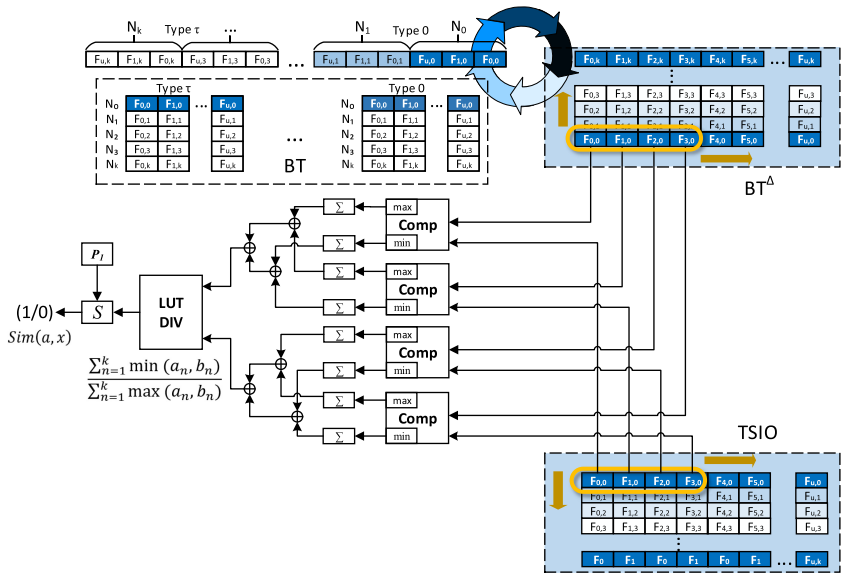

Fig. 3. The core similarity estimation process within ACE-GCN. A time-division-multiplexed sequence scheduling (represented by a cyclical arrow symbol) ensures a constant streaming of τ^{Δ} types from BT to a prefetching buffer BT^{Δ}. Each feature from the tiled SIO passes in synchronization with its counterpart TIO for a fast minimum/maximum classification, then accumulates and iteratively visits the next four pairs until considering the whole feature vector. A Jaccard similarity coefficient is calculated between both accumulators through a LUT divisor with results circumscribed from 0 to 1. Finally, a threshold P_1 determines the absolute similarity of the pair as in Equation (3). As a condition, sub-graph "a" will be highly similar to sub-graph "b" when each and all of their nodes have a similar counterpart in "b"; whenever this condition is not met and any of the nodes in sub-graph "a" cannot find its own individual similar node in "b," the sub-graphs "a" and "b" in question are considered non-similar. In this last case, FSM will skip all features regarding that TIO and jump to the first feature regarding the first vector of the next type; the comparison may start over.

and maintain an equally sequential communication between on-chip and off-chip resources without incurring major complexity or communication issues. The reconfiguration is triggered by a set of pulses from an embedded frequency generator. A resulting depth of 65,535 entries is addressed by 16-bit input; the configuration is instated over a 12 LUT + 672 M20K memory generic framework.

- ***RL, CL, and CNN:*** The two types of neural networks performed within ACE-GCN are fully connected and 1D convolutional. These are driven by a common set of eight parallel MAC units that operate in a fat-tree 47-bit fixed-point format. Their architecture and cycle description are further explained in Figure 4.
  Inspired by flexible-point CNN frameworks like Ristretto [35], we have pre-calculated the potential resulting values by software means and customized the range of the fixed point to avoid larger accuracy loss. The definitive result is a 47-bit register with 1 MSB for the sign, the next 4 MSB for the integer, and 42 LSB for the fractional part.
  The specific neural operation is activated by the top FSM. In the case of **reduction layer (RL)** and **classification layer (CL)**, shallow fully connected layers are performed by accessing specific sets of weights ($FC1W^{\Delta}$, FC2W) and bias in memory.
  The RL purpose is to create the primordial dimensional reduction over the high dimensionality of arriving SIOs. We have designed this layer able to handle the largest datasets in our

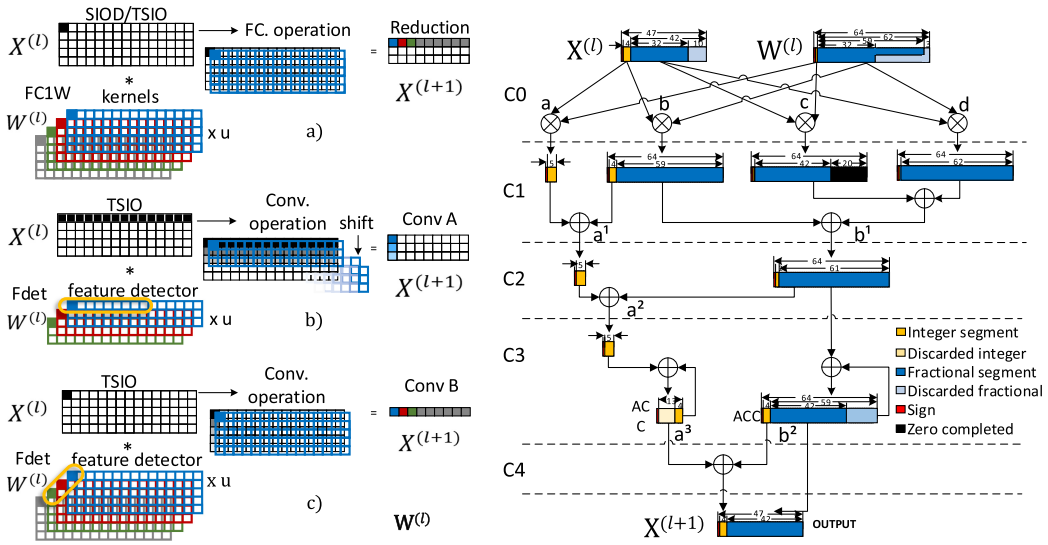

Fig. 4. **(Left)** Overview of the three types of neural networks in ACE-GCN. **(a)** Fully connected convolution, **(b)** 1D convolution with shifting (first intralayer), **(c)** 1D convolution without shifting (second intralayer). Different feature detectors are shown in different colors (blue, red, green, and other gray nets), detector window size per cycle (yellow encirclement), and input features processed (black segments). At the output, other features are calculated if the feature detector window was full, i.e., considers the entire set of detectors (gray segments). Windows are set at eight parallel MACs units. **(Right)** The data-customized MAC cycles: integer and fractional components from features $X^{(l)}$ and kernels $W^{(l)}$ are multiplied and accumulated in a fat-tree top-down configuration, with bit space customized according to the pre-calculated expected range per dataset.

experiments. CL is located at the very end of the process; it realizes the logit classification decision from the definitive SIO embedding. The three modes of neural operations performed by the MACs are graphically explained in Figure 4.

As a CNN, MACs are scheduled to process a 1D CNN using specific feature detectors at different sub-layers (DFET XA, FDET XB) and at different ACE-GCN layers (FDET 2X, FDET 2X). The CNN module produces the linear convolution of the tiled SIO stored at the TSIO sector. Input $X^{(l)}$ has a shape $k \times u$, while the kernel-composed feature detector $W^{(l)}$ has shape $u \times ((k/2)+1)$, from which $(k/2)+1$ is the size of kernels, obtaining the forward propagation $X^{(l+1)}$ described in Equation (1) with shape $u'' \times 1$. CNN results are then normalized and concatenated through **batch normalization (BN)**.

- ***PVE:*** The **prevalence estimator (PVE)** controls which types get stored at BT, limited by τ according to each type counting of successful matching, which we call "events." The PVE organizes the types through the internal table GTL, as per their number of events during the embedding process (P_4 at most). The purpose is to estimate whether a type is worthy of continued storing or, on the contrary, subject to deletion because of irrelevance. This is done to address storing capacity according to memory space availability, as explained in Figure 5.
- ***TC & TM:*** The **type creator (TC)** and the **type matcher (TM)** are functional extensions of PVE. Their purpose is to reflect the changes in GTL at the memory sectors dedicated to BT, BE, and **classification results sector (CLR)** (the last one in the case of TM).
- ***BN:*** This module performs the basic batch normalization equations where complex division and square root operations are pre-calculated and fast available at LUT as in [5]. The

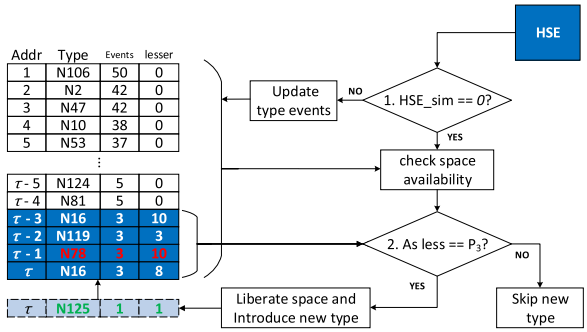

Fig. 5. The graph tracking list (GTL) dynamic. This process tracks and registers the centroids (types) matching with the current sub-graph-in-observance (SIO) and organizes them in the list per number of occurrences or "events." This is done in order to discriminate unimportant centroids based on P_3 storage design parameters.

downside of this approach is the inherent potentiality for accuracy loss due to the utilization of fixed-point format.

4.2 ACE-GCN General Datapath

ACE-GCN is performed in three main stages plus a loop and finalization stage. Stage A performs the gathering and detection of SIOs from the source full graph. Also, it performs an early embedding estimation by exploiting the extra descriptivity that the high dimensionality of raw SIOs may provide. Stage B starts producing the primordial dimensional reduction and tiling abstraction of a raw SIO as provided by the previous stage. Next, it activates the similarity circuit that sorts for appropriate centroids and, if positive, executes in memory and updates such centroid prevalence information.

Finally, stage C occurs whenever stage B fails to locate an appropriate type to produce the SIO embedding estimation. Here, the centroids' storage capacity is evaluated and optimized. Then, if necessary, it activates the neural network to produce a direct convolutional embedding. The purpose is to create new centroids and enrich the SIO comparison basis.

The general datapath overview is graphically presented in Figure 6. and further detailed as follows:

Stage A: Sub-graph abstraction circuit

(1) The source **full graph feature matrix (FGFM)** and **full graph adjacency matrix (FGAM)** are pre-stored within their homonym SDRAM sectors. Each of the N node feature vectors are deployed followed by its respective adjacency vectors. Data is randomly arranged in order to simulate the data-driven environment.
First, the gatherer (GATH) keeps track of the arriving nodes in context to their adjacency vectors, by utilizing a GTL as in Figure 2. Received nodes are immediately transferred back to SDRAM into the **detected SIOs feature vector (DSIOFV)** sector throughout its prefetching buffer DSIOFV$^{\Delta}$. This gathering of graph information proceeds unhalted independently of the rest of the steps until completing the reception of all nodes. Nonetheless, the general process continues to step (2).

(2) Whenever a raw SIO is fully detected at GTL i.e., GATH has received the feature vectors of at least j nodes within one-hop from an specific **reference node (RN)**, the RN feature vector is stored at the SIO reference node feature vector (SIORNFV) sector. Next, it is sent to the HSE.

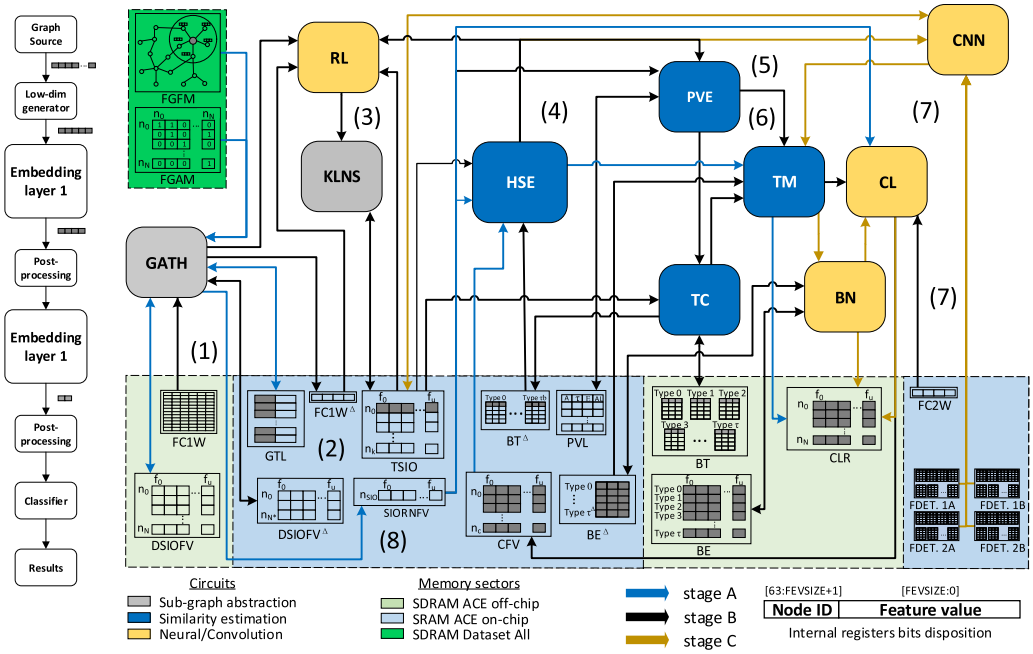

Fig. 6. ACE-GCN core architecture overview. The general process is divided into three main stages. Stage A (blue lines) describes the gathering and detection of SIOs. Stage B (black lines) produces most of the comparison operations. Stage B (golden lines) is activated when realizing the unavoidable convolutional operations. Modules (rounded cases) can intervene on different stages and together form up their main circuits: graph abstraction, similarity estimation, and neural/convolution circuit. Supporting memory sectors (squared cases) are described in light green for SDRAM massive data storage and light blue for on-chip storage, lists, registers, and buffers. Dark green denotes the data-driven emulation environment allocated in SDRAM, and it is not a functional part of ACE-GCN. The eight processing stages are signaled in the figure and explained in Section 4.1. At the left border, a representation of the graph embedding algorithm LGCN [6] motivating our architecture is shown; roughly, it includes a primordial dimensional reduction, two overlapped embedding layers each with post-processing stages, and a final a classification layer.

HSE performs a feature-based similarity coefficient calculation between the SIORNFV and each sample in the **classification feature vector (CFV)** sector at SRAM. If any of the samples matches the current RN (overpass P_1), a positive HSE signal gets the SIO automatically classified on the class pointed by the matching centroid in CFV.
In this context, the first functionality of the TM module updates the node ID (from the RN ID) and the class vector on the CLR and continues to stage B, step (6); otherwise, the process continues normally to stage B, step (3).

Stage B: Similarity estimation circuit

(3) Through GATH, identified SIO RN feature vector and the rest of **neighboring node (NN)** feature vectors are extracted from the DSIOFV sector. Also, all the associated sets of weights are extracted from the **fully connected layer weights (FC1W$^{\Delta}$)** sector. This information is sent directly to the dimensional RL module. All the extraction is done following the NN IDs listed for that detected SIO within GTL.
At each feature arrival, RL starts the reduction operation of the high-dimensional SIO feature vector. For this, the RL processes the now **"tiled" sub-graph-in-observance (TSIO)** from

its homonym sector; this produces a primordial dimensional reduction. Details on the RL itself can be seen in Figure 4 (fully connected layer).
The resulting vectors are evaluated by the KLNS module that realizes a heuristic ranking of the processed features to get a k-limited abstraction [6]. KLNS uses the TSIO sector itself to realize its calculations, ultimately obtaining the wanted TSIO abstraction. Continues to step (4).

(4) The HSE compares and calculates the feature-based similarity coefficient between the TSIO content and each of the types stored at the **bank of types prefetching buffer (BT^{Δ})**. Whenever each type is being considered for comparison, we refer to it as a TIO.
After the evaluation, if a positive HSE signal has been generated, the process continues to step (5); if not, the next TIO is iteratively visited. After parading all the TIOs contained at BT^{Δ} (limited by P_2), if no available TIO ever matched TSIO (no positive HSE signal was generated), then the process continues to stage C, step (6). Both paths are better illustrated in Figure 5.

(5) Having encountered a type matching the TSIO, the PVE increases the similarity events counting within the **prevalence estimator list (PVL)** with regard to the matching TIO.
PVE searches for its reference node ID within the event list, and then the associated number of events is updated. Next, GTL is reorganized from more to fewer events, and types located at the very end of the list get identified "as lesser"; this category may include one or several types (P_3 at most).
In this context, the **type creator (TC)** module is not activated. The TM third functionality gets the embedding vector result that is associated with the matching TIO from the bank of embeddings prefetching buffer (BT^{Δ}), passing through the BN module to the CLR sector, without awaiting results from the actual CNN module. This operation is what we refer to as convolutional embedding estimation. To avoid additional accuracy loss, we do not update CFV when recurring to embedding estimation. Continues to step (8).

Stage C: Neural/Convolution circuit

(6) Since HSE could not find a similar type, PVE must introduce a new one to its GTL (if there is no room, this introduction is ignored and only executes TM tertiary functionality). Every time a request to create a new type is generated, GTL will increase a counter for the element (or elements) identified "as lesser."
P_3 must be reached on this counter for the threatened type to be considered "irrelevant." If so, the list automatically releases that memory space by shifting all the above elements one space up and transfers the node ID of the matching type (that originated the request) to the last space of the list, assuming from now on the category "as lesser" with a single event recorded.
The counter of the previously identified "as lesser" gets reset (unless they were pushed in or out of this category, as newer events arrive). Then, the content of the TSIO sector is passed feature by feature into BT (through BT^{Δ}) by the TC module. Afterward, the tertiary functionality of TM is activated; it transfers the resulting vector produced by the neural module (CNN) to both BE and CLR (through BN). Continues to step (7).

(7) Since there is no previously calculated embedding for the current SIO, the CNN module must produce one. CNN processes the content of the TSIO sector jointly with the **feature detector (FDET)** sectors destined for each l_{DNN} (1-D CNN intralayers) and l_{ACE} (ACE layers) and served consecutively.

Table 3. Synthesis and Benchmarking Equipment Utilized for This Study

ACE-GCN	AWB-GCN	PyG	LGCN
SoC Development Kit	Intel programmable acceleration card D5005	Intel Xeon E5-2680	Intel Xeon Gold 6148
Intel Stratix 10 SX (2800)	(2800) Stratix 10SX FPGA	2.5 GHz	2.4 GHz
330 MHz	330 MHz	Nvidia RTX8000	Nvidia RTX 2080 TI
3,732,480 ALM	3,732,480 ALM	1,395 MHz	1,350 MHz
244 Mb SRAM	244 Mb SRAM		
16 GB DDR SDRAM	32 GB DDR SDRAM		

AWB-GCN and PyG hardware information is directly provided by the study in [7]. For experimental fairness when deploying ACE-GCN and LGCN software execution, we have utilized available equipment that closely matches that utilized by AWB-GCN.

Reutilizing the same TSIO sector as an intermediary layer and switching among the proper FDET, CNN produces both l_{DNN} of the current l_{ACE}, and then the results are processed by the BN module.
As a condition, for an SIO to be allowed to process the next l_{ACE}, it is mandatory that all of its k-largest neighboring nodes have been processed on the current l_{ACE} layer as well (CL does not require this, as it is the last one).
If a specific SIO analysis has not completed every l_{DNN} layer, its last BN embedded representation remains at the CLR sector (by the tertiary functionality of TM, on behalf of the current SIO reference node), waiting for the rest of its NNs to be equally processed.
For processing a new l_{DNN} layer, the FDET sector is switched to the concerning one (1A, 1B, 2A, or 2B). If all l_{DNN} layers concerning that SIO have been completed, CL processes the definitive embedding result with FC2W into the targeted classification vector. Afterward, the tertiary functionality of TM transfers the definitive resulting vector produced by CNN (through BN) to both BE ($^{\Delta}$) and CLR sectors.
In parallel in this step, if the CFV sector has not completed a P_3 quota of feature vectors per class (defined by dataset), and the class detected has not been previously filled up, the current SIO reference node ID (from the SIO reference node feature vector) is stored in the account of its detected class, meeting conditions of step (2).
The reference node ID bits get stored followed by the bits indicating its classification. Continues to step (8).

Stage F: Loop and finalization

(8) The process goes back to step (2) until classifying every N nodes. END.

5 EXPERIMENTAL METHODOLOGY

ACE-GCN is designed at RTL with Verilog, synthesized with Quartus Prime, and paired as a custom logic with Qsys. Nios-II is used only as a facilitator for the Avalon communication protocol with off-chip memory and general SoC-PC data traffic. Latency measurement is obtained by tracking the time of graph inference completion, minus the latency caused by the initial source graph broadcasting (data-driven setup). As an accuracy metric, we use the **area under the curve (AUC)** method and accuracy loss in % based on it. For acceleration measurements, we have calculated the average from 10 readings each under equivalent conditions.

Details on the experimental devices and equipment are shown in Table 3 along with the device specifications presented by AWB-GCN [7]. Parameters and datasets utilized in this work are detailed in Table 4 and Table 5, respectively. ACE-GCN is designed for inference acceleration only and neural kernels are pre-calculated offline with the Glorot initialization technique [11]. Also,

Table 4. List of Parameters Utilized in Our Study and Their Meaning

Symbol	Value	Definition	Symbol	Value	Definition
N	(Dataset)	Number of nodes in graph	τ_3	32,768	Number of types
N^b	(Dataset)	Binary shape of N	τ_2	16,384	"
k	8	k-limited node degree	τ_1	8,192	"
C	(Dataset)	Classes on graph	τ_0	4,096	"
C^b	(Dataset)	Binary shape of C	τ_{-1}	2,048	"
j	(Dataset)	Maximum node degree	τ_{-2}	1,024	"
u^0	(Dataset)	Feature dimensions	τ_{-3}	512	"
u^1	32	"	τ	(Dataset)	BT buffer: 128/256/512
u^2	20	"	P_1	93%	Minimal similarity
u^3	8	"	P_2	8	Samples per class
u^4	16	"	P_3	15	Maximum types "as lesser"
u^5	12	"	P_3^b	4	Binary shape of P3
u^6	8	"	P_4	255	Maximum events per type
u^7	24	"	P_4^b	8	Binary shape of P4

Some parameter values are defined by specific datasets; the rest are fixed design values generated by software pre-training and ablation analysis according to performance requirements.

Table 5. Real-World Graph Datasets Utilized in this Study

Dataset	V	E	Degree	Classes
Cora	2,708	13,264	5	7
CiteSeer	3,327	4.732	4	6
PubMed	19,717	108,365	6	3
NELL	65,755	318,135	5	210
Cont-201	80,595	199,199	4	4
Fome20	108,175	232,645	4	4
D6-6	120,576	146,875	2	5
Reddit	232,965	5,376,619	23	41

According to their sparsity and size: Highly sparse small-sized (Cora, CiteSeer, PubMed), highly sparse medium-sized (NELL, Cont-201), highly sparse large-sized (Fome20, D6-6), and moderately sparse large-sized (Reddit) (Cont-201, Fome20, and D6-6 not included in AWB-GCN; same for LGCN besides NELL). Only for the smaller group we have separately generated smaller customized configurations of ACE-GCN, while the rest of the larger datasets share a single configuration generated with wider specs.

due to the cyclic nature of the phenomena in aim with known, stable, and predictable behavior, we only consider transductive learning in our deployment and experimentation.

We have chosen AWB-GCN as the main comparative reference since it is one of the few proper GCN accelerators published to date. It is also comparable to our study regarding its deployment environment (actual FPGA physical synthesis), selected datasets, aimed inference applications, and concerns on performance/power/resources results.

Having said that, our accelerator greatly differs in nature from AWB-GCN. ACE-GCN is not an open framework hosting multiple GNN models, as in AWB-GCN. However, since its acceleration principles are isolated from the neural component, it can be paired to a diverse range of neural networks.

We have implemented three acceleration approaches and included two reported acceleration results from the literature:

(1) Full ACE-GCN: Full implementation of ACE-GCN on FPGA including similarity estimation circuit plus the neural circuit containing the customized fat-tree fixed-point FC, CL, and CNN cores. Seven τ levels and four dataset super-customized configurations of ACE-GCN are generated.
(2) Standalone (S) mode, basic GCN embedding hardware implementation: GCN embedding only executed with the customized fat-tree fixed-point FC, CL, and CNN cores on FPGA. Absence of the similarity estimation circuit.
(3) LGCN baseline software execution: A pure software Python-based GCN embedding running in a GPU solution. For this segment, we utilize a modified version of the publicly available LGCN open source code [6].
(4) AWB-GCN: Hardware performance and specifications reported by the AWB-GCN study [7] on its full configuration (with rebalancing technique) and 4K PE. To our knowledge (along with its predecessor UWB-GCN [8]) it is the only actual FPGA accelerator, fully implemented and tested on GCN inference, with similar technical target and dataset type to ours.
(5) PyG baseline software execution: The reported CPU-based Python Geometric performance from AWB-GCN study.

We evaluate ACE-GCN under six different contexts as follow:

(1) Acceleration and accuracy loss under different storage levels τ (logarithmic scale): Speed-up obtained by ACE-GCN under seven τ variances (τ_{-2}, τ_{-1}, τ_0, τ_2, τ_1 and extreme variances τ_{-3} and τ_3), its standalone mode implementation, and AWB-GCN (full) regarding PyG software execution at all the datasets available.
(2) Neural circuit utilization under extreme τ variances: To further understand the effect of extreme variances in storage capacity as a graph convolutional operation reliever and introduce the potentiality for parallelism of ACE-GCN under extreme τ selections, we present the progressive utilization levels of the neural circuit throughout the graph embedding process for τ_{-3}, τ_0, τ_3 at all the datasets available.
(3) General on-chip storage demand (logarithmic scale): SRAM memory demand for ACE-GCN τ_3 versus that reported for AWB-GCN baseline and its most conservative design implementation (D(B)), throughout the four basic configurations (NELL, Cont-201, Fome20, D6-6, Reddit sharing the same configuration).
(4) General on-chip area demand: Reports the on-chip area demand of ACE-GCN τ_3 versus AWB-GCN baseline and (D(B)) implementation, throughout the four basic configurations.
(5) Power-gain and energy efficiency: Energy efficiency per dataset is presented by the % of power grain (graph/kJ) of ACE-GCN0 over AWB-GCN and regarding PyG software execution. All datasets included except non-comparable: Cont-201, Fome20, and D6-6.

6 PERFORMANCE ANALYSIS

6.1 Acceleration and Accuracy Loss under Different Storage Levels

In Figure 7 (from left to right) it can be observed that for all cases, adding storage capacity to ACE-GCN leads considerable speed-up. In comparison with any of its τ variances, the acceleration provided by the standalone mode is minimal; this indicates the importance of the estimation circuit over the general performance (except for the extreme lower τ_{-3}, which actually produces slowdown in most cases). The best result of full ACE-GCN at Cora is obtained for τ_3 at around

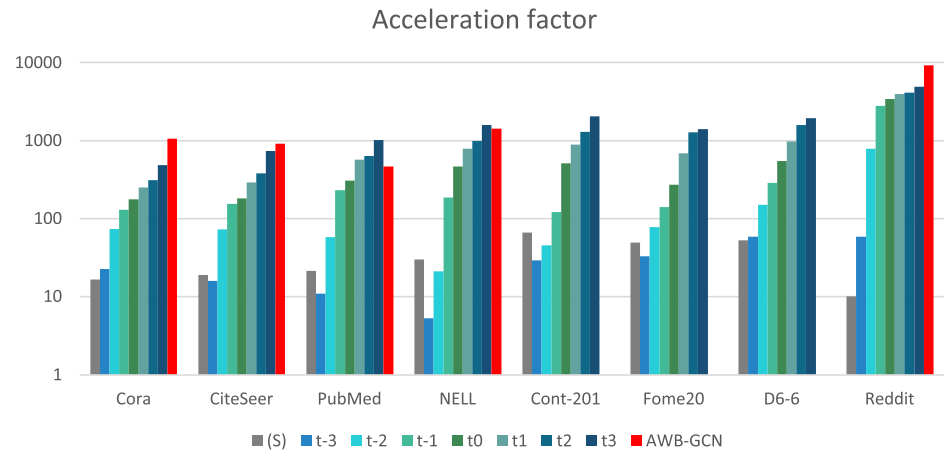

Fig. 7. Report on the acceleration factor of ACE-GCN in terms of x PyG baseline (Log scale) (at ordinate) in standalone mode, seven different levels of τ, and AWB-GCN reported results in red bars (eight groups in total) (at abscissa). Datasets Cont-201, Fome20, and D6-6 not included in AWB-GCN study.

486× PyG; however, this is less than half that reported by AWB-GCN for the same case (1,063× PyG).

Nonetheless, as the graph size increases, the gap between ACE-GCN and AWB-GCN ostensibly decreases. For instance, for CiteSeer, ACE-GCN at τ_3 is only around 0.81× AWB-GCN (183× PyG under AWB-GCN). For PubMed, ACE-GCN at τ_2 already manages to overpass AWB-GCN multiprocessing results by 2.18×, i.e., 551× PyG above AWB-GCN, while at τ_3, ACE-GCN acceleration is around 1.11× AWB-GCN for a still considerable margin of 165× PyG above AWB-GCN. The explanation for these results might be related to the increasing availability of more representative and richer matching events to the bank of types.

Indeed, PubMed dataset sparsity, along with its size and reduced number of classes, ensures a higher probability of events; this effect is in concordance with the relation in Equation (5). Conversely, for similar node degree at smaller graphs, matching events are less likely to happen, resting more frequently on the neural circuit to obtain a proper convolution, thus having acceleration substantially reduced. On the other hand, it can be seen that the NELL acceleration gap with AWB-GCN is no longer so plenteous. Although NELL presents all the requirements to obtain greater acceleration, its 240 classes affect our conservative hardware goals; nevertheless, acceleration results are still distinguishable.

To demonstrate the range capacity of our implementation without incurring additional hardware requirements, we have reutilized the same specs generated for NELL on the rest of the relatively larger datasets, equally capping storage capacity at different τ. In the case of Cont-201, Fome20, and D6-6, datasets has been deliberately chosen to match our acceleration principle at the best, i.e., super-sized, highly sparse graphs with power law distribution and reduced number of classes.

In general, it can be observed that our approach reaches unimpeded acceleration, up to 2,000× PyG at the best case (τ_3) and around 500∼900× for more conservative specs (τ_0). We also observe a more evident logistic growth per τ increase but no vast differences among the three datasets regardless of their exact size. In terms of acceleration, this indicates that for a fixed configuration, the actual dataset size is not so determinant as it is a richer availability of centroids.

The largest dataset in our experiment Reddit has nearly twice the size of D6-6 and 8x its number of classes and exhibits a level of sparsity degree 10x lesser. Still utilizing the same

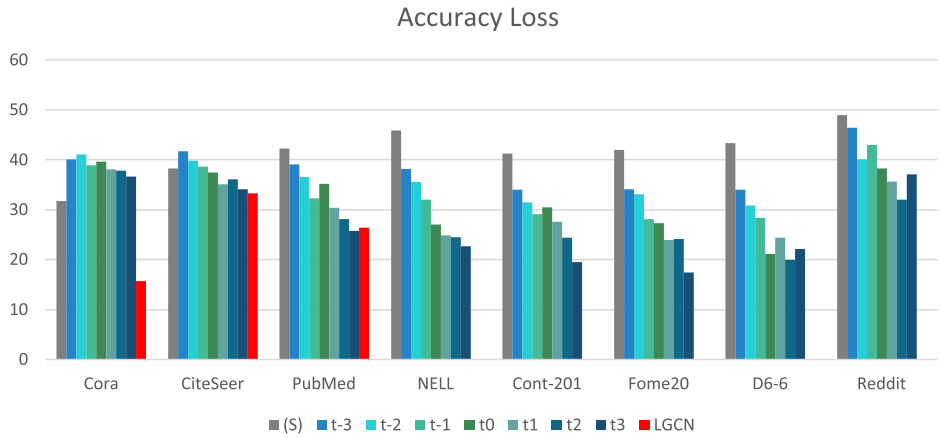

Fig. 8. Report on the accuracy loss of ACE-GCN (100 - absolute accuracy) (at ordinate) in standalone mode, seven different levels of τ, and original LGCN algorithm in red bars (eight groups in total) (at abscissa). Datasets Cont-201, Fome20, and D6-6 were not included in AWB-GCN study.

configuration common to NELL, our accelerator manages to provide acceleration to the task when compared to software execution (around 4, 931× PyG for τ_3). This is, however, 4, 492× PyG less than AWB-GCN for the same scenario. Nonetheless, we also observe other characteristics. For instance, the contribution of the estimation circuit to the performance is even greater this time, especially for the smaller τ. This tendency seems to rapidly slow down, although reaching a plateau that becomes more evident in the logarithmic representation of Figure 7. A direct explanation of this situation might be related to the rather basic memory fetching system utilized by ACE-GCN and likely unrelated to the acceleration principle itself.

The acceleration benefits of introducing convolution estimations do not come without charge. In Figure 8, it can be observed that for the smaller datasets like Cora and CiteSeer, accuracy loss seems to reach as high as 41%, while there is a slight improvement for larger datasets with PubMed and NELL with maximums at around 38∼39%.

Being in essence an approximation technique, implicit-processing-by-association brings an inherent loss in accuracy. In comparison, the fixed-point-based standalone mode of ACE-GCN performs better for Cora and CiteSeer, although it becomes unusable when increasing the graph size at PubMed and NELL (with losses at around 42% and 46%, respectively).

Fortunately, processing CiteSeer and PubMed configurations at τ_3, in general, preempted our accelerator of reaching additional loss and even made it slightly improve accuracy over the LGCN algorithm itself, with a loss of only 34.1% and 25.83%, respectively, and a small gain of 1.62 points over LGCN in the case of PubMed. The explanation for this seems to be in relation to those studied at Equations (6) and (7).

First, as its storage capacity becomes plenteous, our accelerator has access to a wider diversity of potentially matching centroids, granting a more accurate comparison basis during the estimation process. Second, as the dataset size increases and a high sparsity relation is maintained, the intrinsic representativeness of its elements also increases. This eventually produces higher statistical confidence on each of the detected centroids or types.

We can observe the same effect in the larger group: Cont-201, Fome20, and D6-6. While the exact graph size does not produce perceptible accuracy loss differences (which seems to be bounded to around 33∼34%), within each, storing a larger quantity of types tends to improve task accuracy for a considerable margin, with the best-case scenarios at τ_3 from 17.48% to 22.17% approximately.

Table 6. Experimental Results of ACE-GCN in Standalone Mode and Seven τ Variances, Plus AWB-GCN and CPU-Based PyG Reported Results

Platform	Standard	Cora	CiteSeer	PubMed	NELL	Reddit
Intel Xeo E5-2680 (PyG)	Latency (ms) [speedup]	2.51 [1x]	3.66 [1x]	13.97 [1x]	2.28E3 [1X]	2.94E5 [1x]
	Energy efficiency (graph/kJ)	6.68E+03	3.88E+03	1.03E+03	6.99	5.43E-02
AWB-GCN Intel D5005 FPGA	Latency (ms) [speedup]	2.32E-3 [1063x]	4.0e-3 [913x]	3E-2 [466x]	1.6 [1,425x]	31.81 [9242x]
	Energy efficiency (graph/kJ)	3.08E+06	1.93E+06	2.48E+05	4.12E+03	2.09E+02
ACE-GCN (τ_3) Stratix V GX	Latency (ms) [speedup]	5.12E-3 [486.4x]	4.99E-3 [732.5x]	1.37E-2 [1,017.1x]	1.44 [1,582.5x]	5.96E+01 [4931.5x]
	Energy efficiency (graph/kJ)	1.43E+07	1.15E+07	2.76E+06	6.36E+04	3.69E+05
	Accuracy (AUC)	63.33	65.90	74.17	77.31	62.86
ACE-GCN (τ_2) Stratix V GX	Latency (ms) [speedup]	8.07E-3[310.8x]	9.65E-3[379x]	4.73[621.3x]	2.30[990.2x]	7.14E+01 [4116.28x]
	Energy efficiency (graph/kJ)	1.26E+07	9.07E+06	2.41E+06	5.78E+04	2.19E+05
	Accuracy (AUC)	62.18	63.91	71.86	75.52	67.92
ACE-GCN (τ_1) Stratix V GX	Latency (ms) [speedup]	9.96E-3 [250.8x]	1.25E-2 [291.1x]	2.44E-2 [572x]	2.92 [784.8x]	7.53E+01 [3902.42x]
	Energy efficiency (graph/kJ)	1.00E+07	7.94E+06	1.93E+06	3.79E+04	1.30E+05
	Accuracy (AUC)	61.92	64.92	69.59	75.12	64.36
ACE-GCN (τ_0) Stratix V GX	Latency (ms) [speedup]	1.41E-2 [177.3x]	2.02E-2 [180.8x]	4.55E2 [306.4x]	4.86 [469x]	8.58E+01 [3425.3x]
	Energy efficiency (graph/kJ)	6.27E+06	6.06E+06	1.26E+06	1.59E+04	8.08E+04
	Accuracy (AUC)	60.35	62.57	64.81	72.93	61.76
ACE-GCN (τ_{-1}) Stratix V GX	Latency (ms) [speedup]	1.91E-2 [130.8x]	2.36E-2 [154.5x]	5.98E-2 [233.3x]	12.03[187.9x]	1.06E+02 [2757.2x]
	Energy efficiency (graph/kJ)	2.25E+06	1.55E+06	3.96E+05	9.20E+03	3.83E+04
	Accuracy (AUC)	61.06	61.32	67.72	67.98	57.00
ACE-GCN (τ_{-2}) Stratix V GX	Latency (ms) [speedup]	3.38E-2 [74.1x]	4.89E2 [74.7x]	0.24 [58x]	1.08E2 [21.1x]	3.72E+02 [789.14x]
	Energy efficiency (graph/kJ)	1.45E+06	6.19E+05	4.33E+04	1.84E+03	3.13E+03
	Accuracy (AUC)	58.93	60.21	63.48	64.45	59.82
ACE-GCN (τ_{-3}) Stratix V GX	Latency (ms) [speedup]	0.11 [22.7x]	0.23 [15.9x]	1.28 [10.9x]	4.3E2 [5.3x]	5.00E+03 [58.79x]
	Energy efficiency (graph/kJ)	7.11E+05	1.21E+05	1.84E+04	3.37E+02	3.91E+03
	Accuracy (AUC)	59.91	58.32	60.94	61.77	53.56
ACE-GCN (S) Stratix V GX	Latency (ms) [speedup]	8.3E-2 [29.9x]	0.17 [21.5x]	1.53 [9.1x]	3.43E+02 [6.63x]	2.90E+04 [10.11x]
	Energy efficiency (graph/kJ)	8.23E+05	3.48E+05	2.12E+04	1.32E+02	1.30E+02
	Accuracy (AUC)	68.28	61.68	57.75	54.08	51.19

Latency (ms), energy efficiency (graph/kJ), and AUC absolute accuracy (only for ACE-GCN).

When dealing with Reddit, accuracy loss had its maximal at around 46% for τ_{-3} and its minimum at around 32% for τ_2. A better detailed experimental report on this segment is presented in Table 6.

6.2 Neural Circuit Utilization under Extreme τ Variance

As previously seen, there is a relation between the amount of memory resources assigned to the ACE-GCN similarity estimation circuitry and its performance in terms of acceleration and accuracy; i.e., as larger τ are introduced, there is less reliance in the neural circuit. We further analyze this characteristic by observing the progress of the average number of requests to the neural circuit throughout a full graph inference to completion time.

For this instance, we have studied the development of neural circuit utilization under three extreme variations of τ: reference τ_0, negative variation τ_{-3}, and positive variation τ_3. The objective is to reflect the behavior of the accelerator whenever τ is located at extreme points.

Figure 9(a) presents the case for reference τ_0; the progress of neural circuit requests tends to decrease without too many spikes till reaching an almost zero utilization at around 35∼40% for a medium-sized dataset and 70% for larger datasets (% to task completion). At zero point, the types on storage have been greatly refined, and the circuit has been able to assume the following SIO embeddings for the rest of the execution time.

In Figure 9(b), τ_{-3} is extremely low in regard to the reference. In this situation, the accelerator is never totally able to disregard the neural modules. Cora and CiteSeer, being small-sized graphs, get several touches over zero utilization, but they are rapidly forced to forget and restart embedding-similarity operations, producing the observed spikes in the figure. The situation with PubMed and NELL is similar but located even farther above zero, while for the more complex Reddit, utilization bounces generally away from zero utilization.

The lack of enough memory resources produces a distortion on the similarity matching functionality of our accelerator. Finally, with an extremely large τ_3 at Figure 9(c), zero level is quickly achieved for all datasets, hitting 15% and 20% for medium-sized graphs and 30% to 45% for larger

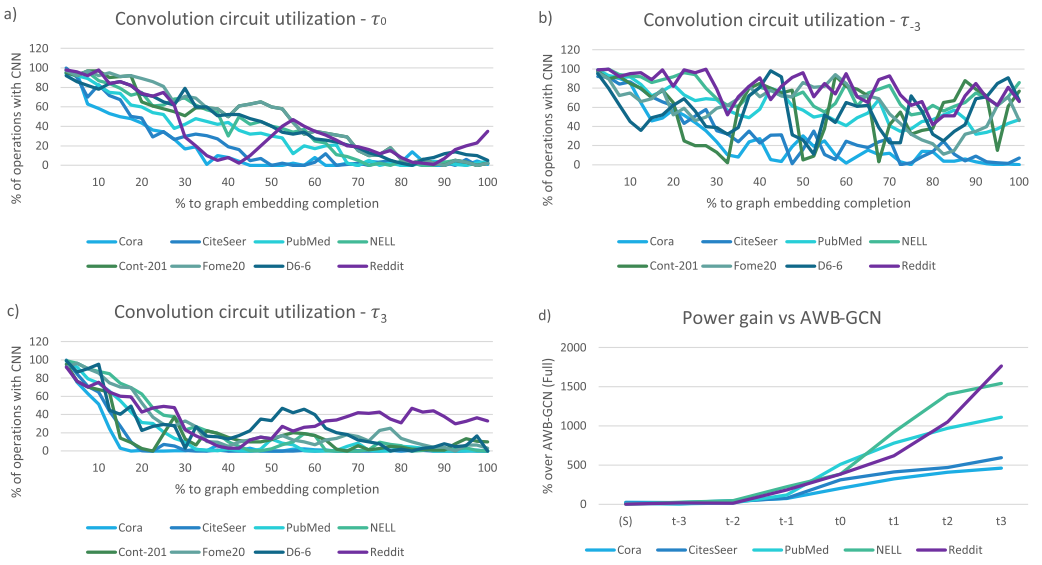

Fig. 9. **(a)** Convolution circuit utilization for τ_0 at all datasets available. The x-axis reports the progress to 100% embedding task completion for the whole graph. The y-axis reports % of convolutional module utilization regarding similarity circuit utilization from total number of operations. **(b)** Convolution circuit utilization forτ_{-3} at all datasets available. **(c)** Convolution circuit utilization for τ_3 at all datasets available. **(d)** Power gain obtained by ACE-GCN over AWB-GCN (y-axis) regarding energy efficiency (in graph/kJ), for the five comparable datasets, throughout seven τ levels (x-axis).

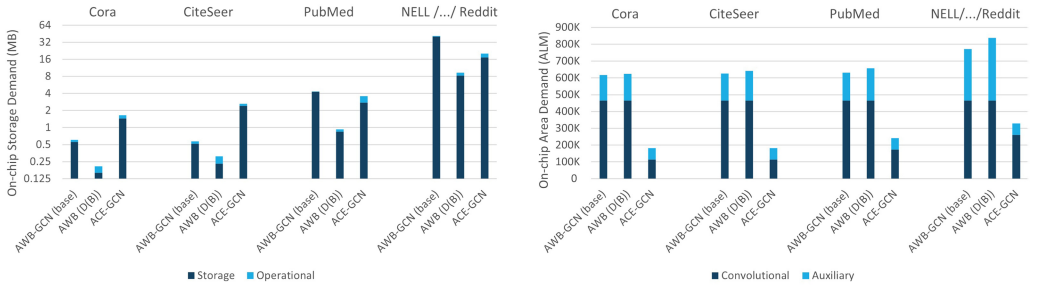

Fig. 10. **(Left)** On-chip storage demand and composition of ACE-GCN (normalized) and AWB-GCN (baseline and design B). Repository and operational composition are differentiated by colors. Dark blue shows the repository memory elements, while light blue shows the operational, control, and communication elements at the four basic configurations. **(Right)** On-chip area demand and composition of ACE-GCN and AWB-GCN (baseline and design B). Dark blue shows the elements directly involved in the convolutional operations, while light blue shows the auxiliary, control, and communication elements at four basic configurations.

datasets. The general convolutional access is also quite smooth, and the spiking restarts seen in τ_{-3} are replaced by (generally) flatter centroid learning curves.

6.3 General On-chip Storage Demand, Area Demand, and Energy Efficiency

With ACE-GCN, our purpose is to transfer computing power from on-chip to off-chip storage as much as possible. However, to keep satisfactory acceleration markers, our approach still makes use of a considerable amount of on-chip memory, as observed in Figure 10(a). We have customized each configuration to fit its dataset as much as possible. It can be seen that for the smaller Cora

and CiteSeer, our on-chip demand is larger than for AWB-GCN and its more conservative variance AWB(D(B)). This relation soon equates for larger PubMed and even underpasses the less efficient AWB baseline, it remains larger to AWB(D(B)) in every case, with a shorter gap for NELL/. . ./Reddit.

In regards to on-chip area demand, and as it is seen in Figure 10(b), ACE-GCN innovative implicit-processing-by-association widely surpasses the more demanding multi-processing approach of AWB-GCN, in the case of ACE-GCN performing with 200K ALMs on average for all datasets, except for NELL/. . ./Reddit, which is around 300K ALM.

In ACE-GCN, our auxiliary element is mostly represented by the similarity estimation circuit, which is less computationally complex than our proper neural resources; besides, it remains fixed for all ACE-GCN size configurations. Our convolutional module is rather a simplistic one, and it is not optimized for parallel operation; however, the similarity circuit is technically independent of the convolutional, and hence it is possible to adapt more optimized versions of it. This could potentially turn into a reduction of ALM for convolutional operations and improved energy efficiency.

Energy efficiency of ACE-GCN is presented in Figure 9(d) by its power gain over AWB-GCN's own reports (in all cases, based on power efficiency results and in reference to PyG CPU execution). The combination of quicker inference times, close on-chip storage demand, and even smaller on-chip area produces an important surplus of power gain with respect to AWB-GCN. This characteristic only increases with the addition of more storage capacity even for the larger and more complex Reddit dataset.

7 CONCLUSIONS

Currently, the trend among the few GCN accelerators published to date tends to approach the natural sparsity of graphs as an issue that strongly challenges performance and resource scalability. With ACE-GCN, we explored the utilization of this very same quality on our behalf by exploiting first-order sub-graph similarity, feature exchangeability, and structure redundancy for graph convolutional embedding.

The experimental part of this work demonstrates several important characteristics of our approach. First, we have successfully managed to transfer GCN embedding computing complexity into storing capacity while keeping competent values of accuracy. For the maximum number of centroids tested, τ_3 = 32,384, we have obtained up to 4,900× PyG baseline (Reddit), and in some cases surpassed AWB-GCN itself by 605× PyG (PubMed) and 157× PyG (NELL), i.e., 2.18× and 1.11× AWB-GCN, respectively.

Depending on the design goals and dataset size, we have obtained accuracy loss values ranging from 40.09% to 22.65%. For the best accuracy result, ACE-GCN even managed to slightly overcome the LGCN algorithm itself, while for the rest, we still consider them to be functional values within the resource-constrained and real-time standards on target.

In other words, if enough memory capacity is granted, our accelerator is able to provide results faster with competent inference accuracy levels. This is a big advantage for data-driven and general in-place embedded systems, which are usually constrained by technology, resources, and power limitations, and in which usually off-chip memory is plentiful regarding on-chip resources.

In this sense, our architecture has strong advantages over multi-processing approaches like AWB-GCN/UWB-GCN and HyGCN. In the case of AWB-GCN, their traditional, explicit, and ultra-parallel processing implementation requires from 600K to 800K on-chip logic elements. ACE-GCN has a chip usage of only 200K to around 300K ALMs (from which a fixed 82K are functionally independent of the neural modules) and only from 7× to 1.2× on-chip memory regarding the AWB-GCN baseline.

This gradual relief of direct neural resource occupation not only provides embedding acceleration and energy efficiency (up to 1,500% of AWB-GCN for NELL dataset) but also can slowly improve accuracy levels, despite a high utilization of hardware approximation techniques. Moreover, it opens the potentiality for further parallelism by releasing occupancy of the neural circuit to additional parallel operations; this comes in handy for the reduced bandwidth requirements of our system, which is limited to around 206 Gbps in the worst-case scenario.

Highly complex but rigid transversal implementations of traditional solutions unavoidably end up producing exponential communication overhead or higher hardware requirements as a surrogate, disregarding a deeper understanding of graph structures themselves. In our case, we use a more simplistic and straightforward approach to graph inference acceleration, fully adapted to the studied characteristics of super-sparse graph, providing excellent speedup, flexible accuracy, relatively low-frequency operation, and a vast energy and resource difference than baseline.

The results of our study indicate that our acceleration principles could be further expanded into more traditional multi-processing architectures for enhanced graph inference, especially when resource/speed trade-off is a major concern.

REFERENCES

[1] Yunsheng Bai, Hao Ding, Song Bian, Ting Chen, Yizhou Sun, and W. Wang. 2019. SimGNN: A neural network approach to fast graph similarity computation. In *Proceedings of the 12th ACM International Conference on Web Search and Data Mining*.

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

[3] Diana Cai, Trevor Campbell, and T. Broderick. 2016. Edge-exchangeable graphs and sparsity (NIPS 2016). *arXiv: Machine Learning* (2016).

[4] L. Chen, J. Hoey, C. Nugent, D. Cook, and Z. Yu. 2012. Sensor-based activity recognition. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 42 (2012), 790–808.

[5] Gianmarco Dinelli, Gabriele Meoni, Emilio Rapuano, Gionata Benelli, and L. Fanucci. 2019. An FPGA-based hardware accelerator for CNNs using on-chip memories only: Design and benchmarking with Intel Movidius neural compute stick. *International Journal of Reconfigurable Computing* 2019 (2019), 7218758:1–7218758:13.

[6] H. Gao, Zhengyang Wang, and S. Ji. 2018. Large-scale learnable graph convolutional networks. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*.

[7] Tong Geng, Ang Li, Runbin Shi, Chunshu Wu, T. Wang, Yanfei Li, Pouya Haghi, Antonino Tumeo, Shuai Che, Steve Reinhardt, and M. Herbordt. 2020. AWB-GCN: A graph convolutional network accelerator with runtime workload rebalancing. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'20)*, 922–936.

[8] Tong Geng, A. Li, T. Wang, Chunshu Wu, Yanfei Li, Antonino Tumeo, and M. Herbordt. 2019. UWB-GCN: Hardware acceleration of graph-convolution-network through runtime workload rebalancing. *ArXiv* abs/1908.10834 (2019).

[9] Xu Geng, Yaguang Li, Leye Wang, Lingyu Zhang, Qiang Yang, Jieping Ye, and Yan Liu. 2019. Spatiotemporal multi-graph convolution network for ride-hailing demand forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 3656–3663.

[10] R. Gera, Lázaro Alonso, B. Crawford, J. House, J. A. Méndez-Bermúdez, T. Knuth, and R. Miller. 2018. Identifying network structure similarity using spectral graph theory. *Applied Network Science* 3 (2018).

[11] Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*.

[12] Lei He. 2019. EnGN: A high-throughput and energy-efficient accelerator for large graph neural networks. *ArXiv* abs/1909.00155 (2019).

[13] D. Hill and B. Minsker. 2010. Anomaly detection in streaming environmental sensor data: A data-driven modeling approach. *Environmental Modelling and Software* 25 (2010), 1014–1022.

[14] Nachiket Kapre. 2015. Custom FPGA-based soft-processors for sparse graph acceleration. In *2015 IEEE 26th International Conference on Application-specific Systems, Architectures and Processors (ASAP'15)*, 9–16.

[15] Anees Kazi, Shayan Shekarforoush, S. Krishna, Hendrik Burwinkel, G. Vivar, K. Kortuem, Seyed-Ahmad Ahmadi, Shadi Albarqouni, and N. Navab. 2019. InceptionGCN: Receptive field aware graph convolutional network for disease prediction. In *IPMI*.

[16] Thomas Kipf and M. Welling. 2017. Semi-supervised classification with graph convolutional networks. *ArXiv* abs/1609.02907 (2017).

[17] Danai Koutra, U. Kang, Jilles Vreeken, and C. Faloutsos. 2014. VOG: Summarizing and understanding large graphs. *ArXiv* abs/1406.3411 (2014).
[18] G. Li, M. Müller, Ali K. Thabet, and Bernard Ghanem. 2019. DeepGCNs: Can GCNs go as deep as CNNs? In *2019 IEEE/CVF International Conference on Computer Vision (ICCV'19)*, 9266–9275.
[19] Qimai Li, Zhichao Han, and Xiao-Ming Wu. 2018. Deeper insights into graph convolutional networks for semi-supervised learning. *ArXiv* abs/1801.07606 (2018).
[20] S. Li, Junwei Huang, Z. Zhang, Jianhang Liu, Tingpei Huang, and Haihua Chen. 2018. Similarity-based future common neighbors model for link prediction in complex networks. *Scientific Reports* 8 (2018).
[21] L. Lu, Y. Liang, Qingcheng Xiao, and Shengen Yan. 2017. Evaluating fast algorithms for convolutional neural networks on FPGAs. In *2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM'17)*, 101–108.
[22] Milena Mihail and Christos H. Papadimitriou. 2002. On the eigenvalue power law. In *Proceedings of the 6th International Workshop on Randomization and Approximation Techniques (RANDOM'02)*. Springer-Verlag, Berlin, 254–262.
[23] Anurag Mukkara, Nathan Beckmann, Maleen Abeydeera, Xiaosong Ma, and D. Sánchez. 2018. Exploiting locality in graph analytics through hardware-accelerated traversal scheduling. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'18)*, 1–14.
[24] J. Nesetril and P. D. Mendez. 2008. Structural properties of sparse graphs. *Electronic Notes in Discrete Mathematics* 31 (2008), 247–251.
[25] M. Newman. 2005. Power laws, Pareto distributions and Zipf's law. *Contemporary Physics* 46 (2005), 323–351.
[26] Eriko Nurvitadhi, Ganesh Venkatesh, Jaewoong Sim, Debbie Marr, Randy Huang, Jason Hock, Yeong Tat Liew, Krishnan Srivatsan, Duncan J. M. Moss, Suchit Subhaschandra, and Guy Boudoukh. 2017. Can FPGAs beat GPUs in accelerating next-generation deep neural networks? In *FPGA'17*.
[27] H. Reittu, Lasse Leskelä, T. Räty, and M. Fiorucci. 2018. Analysis of large sparse graphs using regular decomposition of graph distance matrices. In *2018 IEEE International Conference on Big Data (Big Data'18)*, 3784–3792.
[28] Athanasios I. Salamanis, Dionisis D. Kehagias, C. K. Filelis-Papadopoulos, D. Tzovaras, and G. Gravvanis. 2016. Managing spatial graph dependencies in large volumes of traffic data for travel-time prediction. *IEEE Transactions on Intelligent Transportation Systems* 17 (2016), 1678–1687.
[29] N. Shervashidze, P. Schweitzer, E. V. Leeuwen, K. Mehlhorn, and K. Borgwardt. 2011. Weisfeiler-Lehman graph kernels. *Journal of Machine Learning Research* 12 (2011), 2539–2561.
[30] L. Shi, Yifan Zhang, Jian Cheng, and H. Lu. 2019. Two-stream adaptive graph convolutional networks for skeleton-based action recognition. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR'19)*, 12018–12027.
[31] Adam Silberstein, Gregory Filpus, K. Munagala, and Jun Yang. 2007. Data-driven processing in sensor networks. In *CIDR*.
[32] Petar Velickovic, Guillem Cucurull, A. Casanova, A. Romero, P. Liò, and Yoshua Bengio. 2018. Graph attention networks. *ArXiv* abs/1710.10903 (2018).
[33] T. Wang, Tong Geng, Ang Li, Xi Jin, and M. Herbordt. 2020. FPDeep: Scalable acceleration of CNN training on deeply-pipelined FPGA clusters. *IEEE Transactions on Computers* 69 (2020), 1143–1158.
[34] Long Wen, X. Li, Liang Gao, and Y. Zhang. 2018. A new convolutional neural network-based data-driven fault diagnosis method. *IEEE Transactions on Industrial Electronics* 65 (2018), 5990–5998.
[35] Yuxuan Xie, B. Liu, Lei Feng, Xi-Peng Li, and Danyin Zou. 2020. A FPGA-oriented quantization scheme for MobileNet-SSD. (2020).
[36] Keyulu Xu, Weihua Hu, J. Leskovec, and S. Jegelka. 2019. How powerful are graph neural networks? *ArXiv* abs/1810.00826 (2019).
[37] Mingyu Yan, L. Deng, X. Hu, Ling Liang, Yujing Feng, Xiaochun Ye, Z. Zhang, D. Fan, and Yuan Xie. 2020. HyGCN: A GCN accelerator with hybrid architecture. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA'20)*, 15–29.
[38] Rex Ying, Zhaoyu Lou, Jiaxuan You, Chengtao Wen, A. Canedo, and J. Leskovec. 2020. Neural subgraph matching. *ArXiv* abs/2007.03092 (2020).
[39] Ting Yu, Haoteng Yin, and Zhanxing Zhu. 2018. Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. In *IJCAI*.
[40] Jialiang Zhang, Soroosh Khoram, and J. Li. 2017. Boosting the performance of FPGA-based graph processor using hybrid memory cube: A case for breadth first search. In *FPGA'17*.
[41] Muhan Zhang and Yixin Chen. 2018. Link prediction based on graph neural networks. *ArXiv* abs/1802.09691 (2018).
[42] Qikui Zhu, B. Du, and P. Yan. 2019. Multi-hop convolutions on weighted graphs. *ArXiv* abs/1911.04978 (2019).

Received September 2020; revised April 2021; accepted June 2021