Computing Graph Neural Networks: A Survey from Algorithms to Accelerators

Sergi Abadal, Akshay Jain, Robert Guirado, Jorge López-Alonso, Eduard Alarcón

***Abstract*—Graph Neural Networks (GNNs) have exploded onto the machine learning scene in recent years owing to their capability to model and learn from graph-structured data. Such an ability has strong implications in a wide variety of fields whose data is inherently relational, for which conventional neural networks do not perform well. Indeed, as recent reviews can attest, research in the area of GNNs has grown rapidly and has lead to the development of a variety of GNN algorithm variants as well as to the exploration of groundbreaking applications in chemistry, neurology, electronics, or communication networks, among others. At the current stage of research, however, the efficient processing of GNNs is still an open challenge for several reasons. Besides of their novelty, GNNs are hard to compute due to their dependence on the input graph, their combination of dense and very sparse operations, or the need to scale to huge graphs in some applications. In this context, this paper aims to make two main contributions. On the one hand, a review of the field of GNNs is presented from the perspective of computing. This includes a brief tutorial on the GNN fundamentals, an overview of the evolution of the field in the last decade, and a summary of operations carried out in the multiple phases of different GNN algorithm variants. On the other hand, an in-depth analysis of current software and hardware acceleration schemes is provided, from which a hardware-software, graph-aware, and communication-centric vision for GNN accelerators is distilled.**

I. Introduction

Machine Learning (ML) has taken the world by storm and has become a fundamental pillar of engineering due to its capacity to solve extremely complex problems, to detect intricate features in oceans of data, or to automatically generate alternatives that outperform well-engineered, well-known, carefully optimized solutions. As a result, the last decade has witnessed an explosive growth in the use of Deep Neural Networks (DNNs) in pursuit of exploiting the advantages of ML in virtually every aspect of our lives [1]: computer vision [2], natural language processing [3], medicine [4] or economics [5] are just a few examples.

However, and in spite of its all-pervasive applicability and potential, it is well-known that not all neural network architectures fit to all problems [6]. DNNs take the input data and attempt to extract knowledge taking into account the inductive bias that the connection architecture of the DNN generates. This, in essence, means that the number of DNN layers and their pre-assumed connections determines its suitability to certain tasks. For instance, by not making any assumption on the structure of the data, conventional fully-connected neural networks are able to master a wide range of tasks at the cost of being less efficient in general than other DNNs [7]. In contrast, techniques such as Convolutional Neural Networks (CNNs) or Recursive Neural Networks (RNNs) are biased towards extracting knowledge from the locality and temporal sequentiality of data. This makes them a better fit for specific tasks such as image recognition or treatment of temporal signals, yet incapable of efficiently handling data with arbitrary structures [8].

In light of the above, there has been a recent interest in deep learning techniques able to model graph-structured data [6], [9], [10]. This structure is inherent to a plethora of problems in the field of complex systems in general, and applicable to particular fields such as communication networks where the topology and routing decisions determine its performance [11], synthetic chemistry where molecular structures determine the compound properties [12], social networks where emergent behavior can arise through personal relations [13], or neuroscience where specific connections between neuron types and brain areas determine brain function [14], among many others.

Graph Neural Networks (GNNs) are a set of connectivity-driven models that, since the late 2000s, have been addressing the need for geometric deep learning [15], [16]. In essence, GNNs adapt their structure to that of an input graph and, through an iterative process of aggregation of information across vertices, capture the complex dependences of the underlying system. This allows to predict properties for specific nodes, connections, or the graph as a whole, and generalize to unseen graphs. Due to these powerful features, many relevant applications such as molecule property prediction [17], recommender systems [18], natural language processing [19], traffic speed prediction [20], critical data classification [21], computer vision [22], resource allocation in computer networks [23], already utilize GNNs to accomplish their tasks.

For all these reasons, recent years have seen a rapid increase in research activity in the field of GNNs (see Fig. 7). Specifically, intense efforts are being directed towards improving the efficiency of algorithms, especially for large graphs, and towards demonstrating their efficacy for the aforementioned application areas. The interested reader will find multiple reviews of the state of the art in GNN algorithms and applications in the literature [6], [9], [10], [24]–[27], most of which we highlight and briefly analyze in Table I. Other key aspects relevant or adjacent to GNNs such as network embedding [28], graph attention models [29], or network structure inference [30] have also received a comprehensive review.

As we will see along this paper, however, less attention has been placed on the efficient processing of such new type of

Sergi Abadal, Akshay Jain, Robert Guirado, Jorge López-Alonso, Eduard Alarcón are with NaNoNetworking Center in Catalunya (N3Cat) at Universitat Politècnica de Catalunya (UPC), Barcelona, Spain. Akshay Jain is the corresponding author (Email: akshay.jain@upc.edu).

TABLE I: Background literature: surveys about GNNs (first block) and including GNNs (second block)

Study [Reference] (Year)	Contributions
Relational inductive biases, deep learning, and graph networks [6] (2018)	• Presents the idea of a graph network as a generalization of GNNs with building blocks • Encompasses well-known models, such as fully connected, convolutional and recurrent networks.
Graph Neural Networks: A Review of Methods and Applications [24] (2018)	• Presents a survey of the various GNN models • Provides a discussion of the applications where GNNs can be utilized, and also provisions a taxonomy • Proposes open research problems, such as dynamicity and scalability in GNNs
A Comprehensive Survey on Graph Neural Networks [25] (2020)	• Overviews of GNNs in data mining and machine learning areas • Provisions a taxonomy for GNN models • Details the application areas of GNNs • Presents potential research directions, such as in scalability, dynamicity of GNNs, etc.
Deep Learning on Graphs: A Survey [10] (2020)	• Provides a discussion on graph versions of recurrent and convolutional networks, autoencoders, reinforcement-learning and adversarial methods • Presents the various application areas and future research directions for deep learning methods
Graph Neural Networks Meet Neural-Symbolic Computing: A Survey and Perspective [26] (2020)	• Elaborates the relationship between GNNs and Neural-Symbolic Computing • Develops multiple GNN models with the perspective of being applied to Neural-Symbolic computing
Geometric Deep Learning: Going beyond Euclidean data [9] (2017)	• Proposes Geometric Deep Learning as an umbrella term for models that operate on non-euclidean dataset representations, including GNNs. • Within GNNs, provides a thorough review of convolutional models
Representation Learning on Graphs: Methods and Applications [27] (2017)	• Reviews the advancements in the area of representation learning on graphs • Primary focus is on the network embedding methods

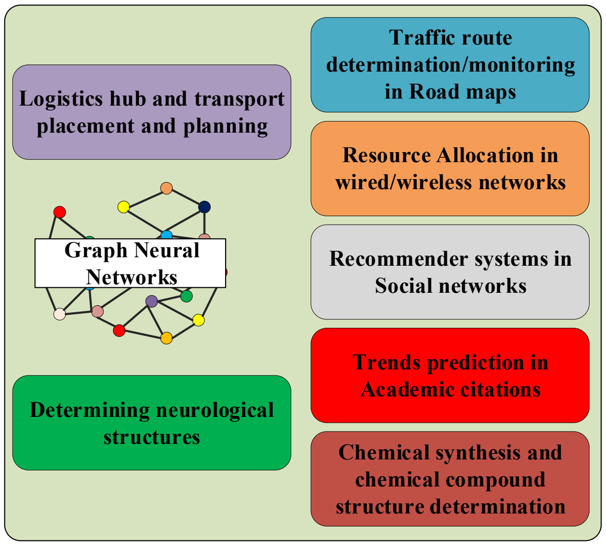
Traffic route determination/monitoring in Road maps
Logistics hub and transport placement and planning
Resource Allocation in wired/wireless networks
Graph Neural Networks
Recommender systems in Social networks
Trends prediction in Academic citations
Determining neurological structures
Chemical synthesis and chemical compound structure determination

Fig. 1: Graph Neural Networks (GNNs) as enablers of a plethora of applications in fields that hinge on graph-structured data.

neural networks. While the issue has already been investigated in significant depth for CNNs or RNNs [31]–[36], GNN processing remains largely unexplored. This is because GNNs are relatively novel and pose unique computing challenges, including the need to (i) support both dense and extremely sparse operations, (ii) adapt the computation to the specific GNN algorithm variant and the structure of the graph at hand, and (iii) scale to very large graphs to realize its potential in certain applications.

In response to the challenges of GNN computing, several works have surfaced that attempt to improve the performance and efficiency of GNNs either from a software perspective, i.e. adapting the operations to better match the capabilities of CPUs or GPUs [37]–[39]; or from a hardware perspective, i.e. designing custom processors tailored to the demands of GNNs [40]–[43]. Also, research in acceleration of sparse/irregular tensors may prove useful in GNNs [44]. However, from the lack of a comprehensive analysis in recent surveys and reviews [6], [9], [10], [24]–[27], we infer that the field of GNN processing is still in its infancy.

This paper aims to bridge this gap by presenting, for the first time, a review of the field of GNNs from the perspective of computing. To that end, we make the following contributions as summarized Fig. 2: we start by providing a comprehensive and tutorial-like description of the fundamentals of GNNs, trying to unify notation. Then, using a Knowledge Graph (KG) approach, we chart the evolution of the field from its inception to the time of this writing, delving into the duality between GNN algorithms (seeing them as learning systems) and their associated computation (seeing them as sets of matrix multiplications and non-linear operations). From that analysis, we identify GNN computing as a nascent field. We finally focus on the computation aspect and provide an in-depth analysis of current software and hardware acceleration schemes, from which we also outline new potential research lines in GNN computing. To the best of the authors' knowledge, this is the first work providing a thorough review of GNN research from the perspective of computation, charting the evolution of the research area and analyzing existing libraries and accelerators.

The rest of this paper is organized as follows: In Section II, we discuss the basics of the GNNs. Section III presents the evolution of the research area from multiple perspectives. In Section IV, we expose the emergent area of GNN accelerators, summarizing recent works and elaborating upon the existing challenges and opportunities. Next, in Section V, we present our vision for the architectural design of GNN accelerators with a focus on internal communication requirements. We conclude this paper in Section VI.

II. Fundamentals of GNNs

In this section, we discuss the basics of GNNs, wherein we not only discuss their building blocks, but also detail the operational aspects and notations based on the ongoing research in the scientific community.

A. Notation

We first describe the main notation for GNNs as summarized in Table II. Let a graph $G = (V, E)$ be defined by a set of vertices V, and a set of edges E that connect some of the vertices in V together. In particular, each vertex $v \in V$ has a neighbourhood set $N(v)$ determined by the edges connecting it to other vertices. Further, each vertex v contains a vertex feature representation h_v, and each edge $e \in E$ contains an edge feature representation g_e. The vertex or edge feature representations are generally one-dimensional vectors containing the scalar attributes that define them. Similarly, the graph may be associated to an global feature representation y containing graph-wide attributes. For example, in a social networking graph, vertices might be users with attributes such as encoded name or location, whereas the edges might be the interaction between two users such as comments/likes on a picture. Graph-wide features may be the number of users living a certain area or voting a certain political party.

A GNN is essentially an algorithm that calculates a set of output feature representations for the vertices h_v, edges g_e, and complete graph y, respectively. Following with the example above, for targeting ads in a social network, output features of a vertex could be the probability of being interested in cars. It can thus be observed that, as in any other neural network, the dimensionality of the output feature vectors will be generally different than that of the input.

As we will see in Section II-B, a GNN is divided in multiple layers. In each layer $l \in [1, L]$, there is an edge aggregation function $\rho_E^{(l)}$ and a node aggregation function $\rho_V^{(l)}$, as well as an edge combination function $\phi_E^{(l)}$ and a node combination function $\phi_V^{(l)}$. The combination functions may be neural networks involving matrices of weights $W_E^{(l)}$ and $W_V^{(l)}$ that are generally common to all edges and nodes, respectively. The outputs of an arbitrary intermediate layer l, given by its combination function, are hidden feature vectors $h_v^{(l)}$ and $g_e^{(l)}$. At the end of the GNN, besides obtaining the output node and edge feature vectors, h_v^L and g_e^L, there are global aggregation and combination functions ρ_G and ϕ_G, respectively, that provide final global output vector $\hat{y}$. We describe the order of these operations in the following sections.

We finally note that, due to the emergence of GNNs, aggregation and combination functions have taken different names in the literature. In an attempt to unify the notation, some equivalences are listed in Table III.

TABLE II: Graph Representation Notations

V	Set of vertices of the graph
E	Set of edges of the graph
$N(v)$	Set of neighbours of vertex v
L	Number of GNN layers
$h_v, h_v^{(l)}, h_v^L$	Input, hidden, output feature vector of vertex v
$g_e, g_e^{(l)}, g_e^L$	Input, hidden, output feature vector of edge e
y	Output global vector
$\rho_V^{(l)}, \rho_E^{(l)}$	Node and edge aggregation functions of layer l
$\phi_V^{(l)}, \phi_E^{(l)}$	Node and edge combination functions of layer l
$W_V^{(l)}, W_E^{(l)}$	Node and edge weight matrices of layer l

B. General Structure

Fundamentally, a GNN is an algorithm that leverages the graph connectivity to learn and model the relationships between nodes. Through an iterative process that depends on the graph structure, the GNN takes the input edge, vertex, and graph feature vectors (representing their known attributes) and transforms them into output feature vectors (representing the target predictions).

In general, the GNN operation is as shown in Fig. 3 and contains the following steps:

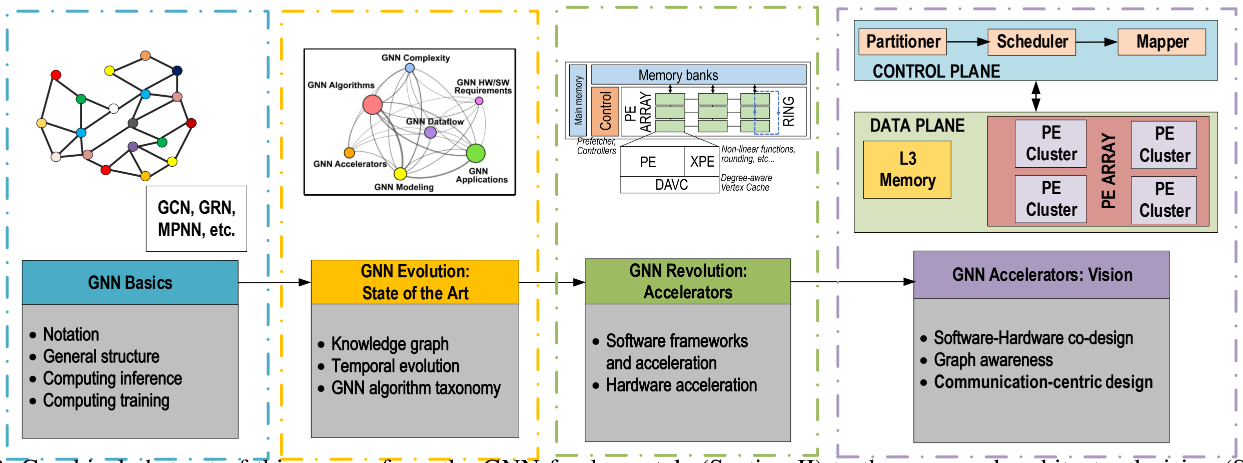

Fig. 2: Graphical abstract of this survey from the GNN fundamentals (Section II) to the proposed architectural vision (Section V).

TABLE III: Aggregate – Combine functions: Homogenized Nomenclature

Aggregation	Combination	Ref.
Local transition	Local output	[16]
Aggregators		[45]
Aggregation	Update	[6]
Message + Aggregate	Update	[46]
Message	Update	[12]
Message	Update	[47]
Message, reduce	Update	[48]
Scatter + ApplyEdge + Gather	ApplyVertex	[38]
Aggregation	Feature extraction + update	[40]
Gather + Reduce	Transform + Activate	[49]
Aggregation	DNN computation	[43]
Aggregation	Embedding	[42]
Aggregate	Combine	[41]
Aggregation	Combination	[50]

1) **Pre-processing:** this is an initial and optional step generally done offline that can transform the input feature vectors and graph structure representation through a precoding process. This may be used to sample the graph, reduce the algorithm complexity, or encode the feature vectors, among others [10], [45], [51].
2) **Iterative updates:** After the pre-processing, the feature vectors of each edge and vertex are updated via the aggregate–combine functions iteratively. To update the edges, attributes from the edge itself, the connected vertices, and the graph are *aggregated* into a single set and *combined* to yield a new edge feature vector. Similarly, updating the vertices implies *aggregating* the feature vectors from neighboring vertices $N(v)$ and *combining* them to obtain a new feature vector. Note that each step or *layer* updates each edge and vertex with information coming from neighbours located at a single hop. Thus, the iterative process allows to gradually account for relations of increasingly distant nodes and edges.
3) **Decoding or readout:** if the graph has a global feature vector, it is updated once after the edge and node updates are completed. The final output is either an edge/node embedding, which is a low dimensional feature vector that represents edge- or node-specific information, or a graph embedding summarizing the information about the entire output graph instead.

As in any other neural network, the GNN processing depends of its architecture. GNNs are basically divided into *layers*, where each layer corresponds to one of the iterations in the update process described above. The larger the number of the layers, the further information from a given node can propagate to another node in the graph. The precise number of required layers thus depends on the application and on how relevant are the relations among *distant* nodes. In general, an excessive amount of layers can lead to over-fitting of models [52] and thus, this number is usually kept low.

In each of the layers, information flows between vertices using an *aggregation* function and feature vectors are updated via the *combination* function after aggregation. The aggregation and combination functions for edges and vertices are determined by the particular GNN algorithm depending on the specific relation to be learnt. As we will see in Section III-D, Table VI, there is a wide variety of such functions ranging from simple averaging or sum for aggregations to different types of neural networks, with their own weighted sums and non-linear activation functions, for combinations [12]. The operations may vary across layers and differ between edges, vertices, or global updates. However, the structure is often simplified by (i) sharing the same operation across layers and (ii) removing or considering trivial combination functions for the updates of edges or nodes. These simplified types are intuitively represented in Fig. 4 of [6].

In summary, we can understand GNNs as a collection of neural networks working over a graph's connectivity. In the scope of each layer, we have up to two neural networks with learnable weights that determine the combination of edges and vertices, respectively. In the scope of the whole GNN, we have a neural network with learnable weights that determines the global update. The way these operations take place for inference and training is depicted next.

C. Computing GNN Inference

Algorithm 1 shows a pseudo-code describing GNN inference. The algorithm may take as inputs the feature vectors of the edges, vertices, and graph; or initialize them. We can see how the execution is divided into layers (line 9) and, within each layer, each and every edge is updated in parallel by aggregating its own feature vector with those of the connected vertices (line 11). Each and every vertex is also updated in parallel by aggregating the feature vectors of its neighbours with itself (line 15). The aggregated edges and vertices are transformed via combination functions (lines 13 and 17), which can be neural networks as we see in Section III-D. Following the completion of the iterative process, a readout is performed using the corresponding function, which may again possibly be a neural network (line 18).

In particular, the algorithm considers that for an arbitrary layer $l \in [1, L]$, edge transformation occurs as

$$\text{AGGR: } b_e^{(l)} = \rho_E^{(l)}(\{g_e^{(l-1)}, h_u^{(l-1)} : u \in N(e)\}), \tag{1}$$

$$\text{COMB: } g_e^{(l)} = \phi_E^{(l)}(\{b_e^{(l)}\}), \tag{2}$$

so that the aggregation of edges ρ_E takes the feature vectors of the edge itself e, g_e and the vertices at its endpoints, h_u with $u \in N(e)$, for the previous layer $l-1$. The combination ϕ_E uses this aggregation as input [53].

A similar reasoning applies to the aggregation and combination of vertices

$$\text{AGGR: } a_v^{(l)} = \rho_V^{(l)}(\{h_v^{(l-1)}, h_u^{(l-1)} : u \in N(v)\}), \tag{3}$$

$$\text{COMB: } h_v^{(l)} = \phi_V^{(l)}(\{a_v^{(l)}\}). \tag{4}$$

The equations indeed describe how $a_v^{(l)}$ is calculated as the aggregation of the feature vectors from the nodes that are neighbours to v, from the previous layer $l-1$, and how the feature vector of layer l is calculated using the aggregation $a_v^{(l)}$ as input.

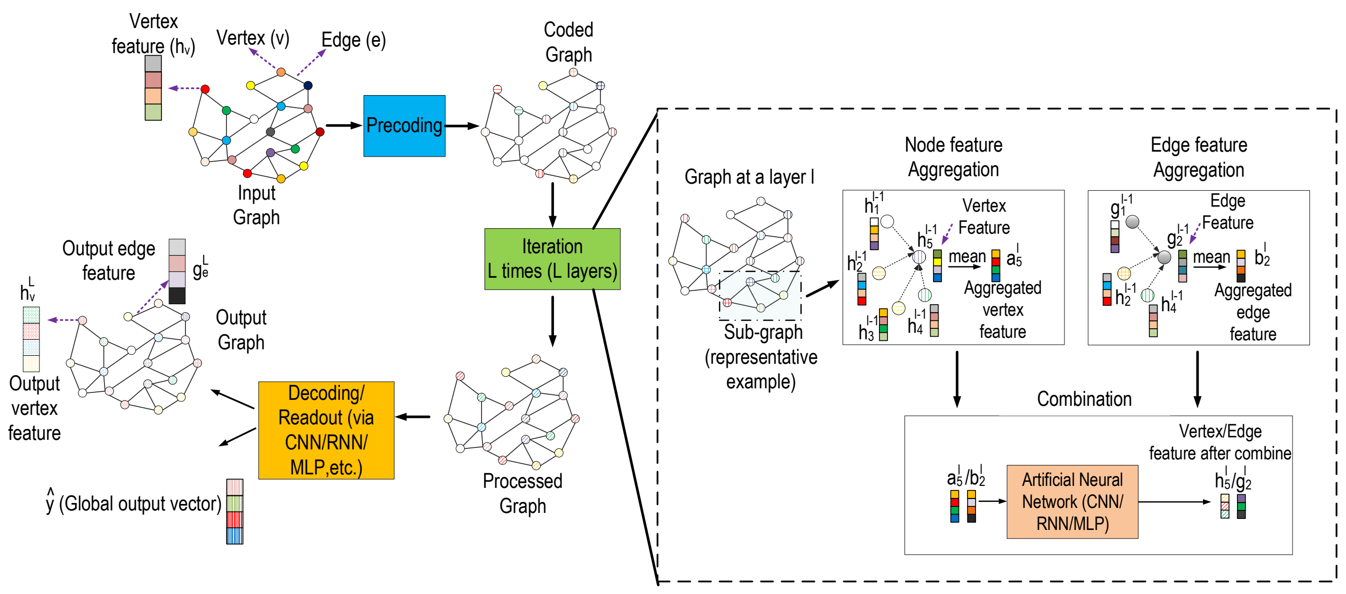

Fig. 3: GNN execution stages during inference: pre-coding, iterative process, and readout.

Algorithm 1 GNN Operations in Inference

```
1:  procedure GNNOPERATOR
2:      $L \leftarrow$ Number of layers in the GNN
3:      $V \leftarrow$ Set of nodes in graph $G$
4:      $E \leftarrow$ Set of edges in graph $G$
    Initialize Nodes and Edges:
5:      for $v \in V$ do
6:          $h_v^0 \leftarrow [x_v,0,\dots,0]$
7:      for $e \in E$ do
8:          $g_e^0 \leftarrow [z_v,0,\dots,0]$
    GNN Layered processing:
9:      for $l = 1$ to $L$ do
    Edge processing:
10:         for $e \in E$ do
11:             $b_e^{(l)} = \rho_E^{(l)}(\{g_e^{(l-1)}, h_u^{(l-1)} : u \in N(e)\})$
12:         for $e \in E$ do
13:             $g_e^{(l)} = \phi_E^{(l)}(\{b_e^{(l)}\})$
    Node processing:
14:         for $v \in V$ do
15:             $a_v^{(l)} = \rho_V^{(l)}(\{h_v^{(l-1)}, h_u^{(l-1)} : u \in N(v)\})$
16:         for $v \in V$ do
17:             $h_v^{(l)} = \phi_V^{(l)}(\{a_v^{(l)}\})$
    Readout:
18:     $\hat{y} = \phi_G(\rho_G(\{h_v^L, g_e^L : v, e \in G\}))$
```

Lastly, following the convergence of the inference process, a final readout function is applied to obtain the output feature vector as

$$\hat{y} = \phi_G(\rho_G(\{h_v^L, g_e^L : v, e \in G\})), \tag{5}$$

which may involve aggregation and combination of feature vectors from edges and vertices of the whole graph, and from the last iteration L.

Algorithm 1 hinges on the general assumption that aggregation and combination functions are (i) invariant to permutation, since there does not exist any implicit order in a graph structure, unless some node feature indicates such an order; and (ii) invariant to the number of input nodes, since the degree of nodes may vary widely across the graph [6]. This implies, as shown in the algorithm, that the functions within a layer can be applied simultaneously first to all edges and then to all vertices. Further, the order between aggregation and combination can be switched if the aggregation function is linear [40]. However, the order of layers needs to be preserved.

To exemplify the computation occurring in inference, top charts of Figure 4 represent the layers of a simple GNN with vertex aggregation and combination only. We show the operations from the perspective of node 1, although all nodes would be realizing the same computations concurrently. We illustrate how the graph connectivity drives the aggregation from nodes 2, 3, and 6 into node 1, and that combination reduces the length of the feature vector through the weight matrices $W^{(1)}$. The second layer repeats the exact same sequence, again reducing the length of the feature vector, this time through a different weight matrix $W^{(2)}$.

Message Passing Equivalence: We note that, similarly to the aggregate–combine terminology, notation relative to GNN algorithms is diverse in the literature. A notable example is that of Message Passing Neural Network (MPNN) [6], which describes the aggregations as message passing functions $M(\cdot)$, the combinations as update functions $U(\cdot)$, or the layers as time steps. Table IV illustrates the equivalence between the MPNN formulations and the corresponding generic formulations from Eqs. (1)-(5).

D. Computing GNN Training

Aggregation, combination, and readout functions can be neural networks, and hence, need to be trained before applying

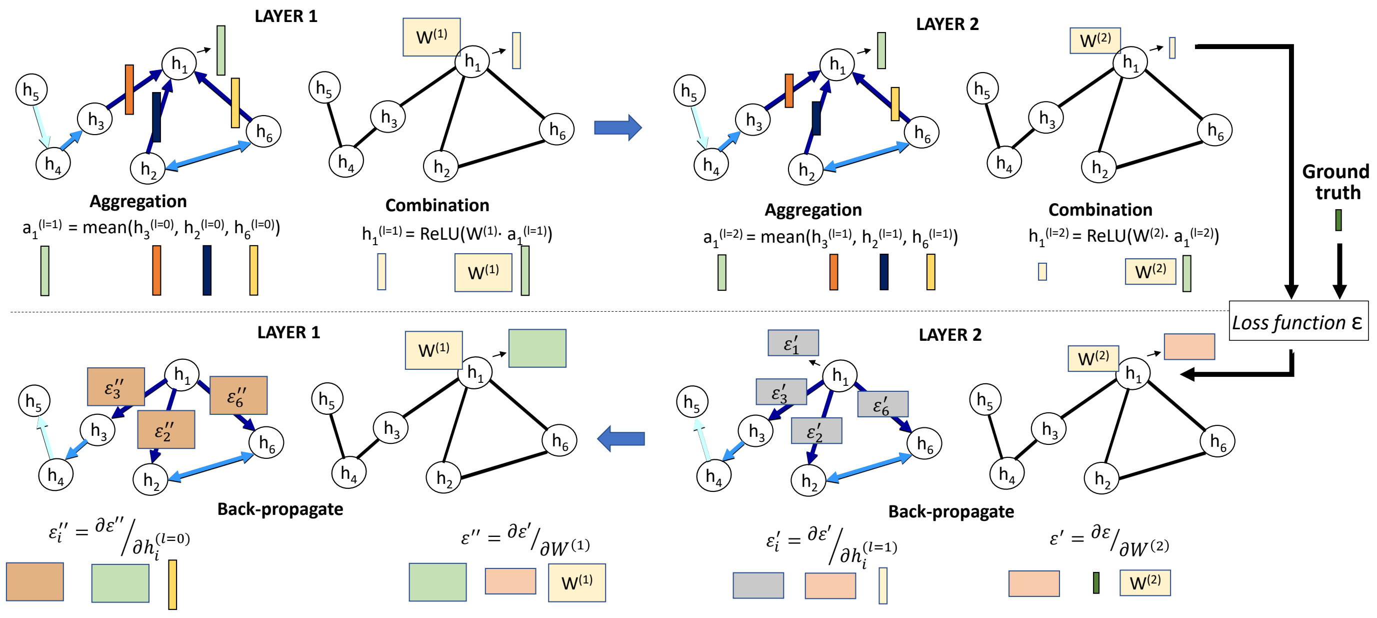
LAYER 1
Aggregation
$a_1^{(l=1)} = \text{mean}(h_3^{(l=0)}, h_2^{(l=0)}, h_6^{(l=0)})$
Combination
$h_1^{(l=1)} = \text{ReLU}(W^{(1)} \cdot a_1^{(l=1)})$
LAYER 2
Aggregation
$a_1^{(l=2)} = \text{mean}(h_3^{(l=1)}, h_2^{(l=1)}, h_6^{(l=1)})$
Combination
$h_1^{(l=2)} = \text{ReLU}(W^{(2)} \cdot a_1^{(l=2)})$
Ground truth
Loss function ε
Back-propagate
$\varepsilon_i'' = \partial\varepsilon'' / \partial h_i^{(l=0)}$
$\varepsilon'' = \partial\varepsilon' / \partial W^{(1)}$
$\varepsilon_i' = \partial\varepsilon' / \partial h_i^{(l=1)}$
$\varepsilon' = \partial\varepsilon / \partial W^{(2)}$

Fig. 4: Example of computation in a sample GNN with node-level aggregation in inference (top left to top right) and training (bottom right to bottom left). The GNN has two layers, mean as the aggregration operator, and weighted ReLu for the combination. We show operations for node 1 only.

TABLE IV: Equivalence between general and Message Passing Neural Network (MPNN) formulations

General	MPNN	Comments
l	t	Layer, time step, or epoch
v	v	Node or vertex of interest
$u \in N(v)$	$w \in N(v)$	Node within the neighboring set $N(v)$ of node v
$h_u^{(l)}$	h_w^t	Feature vector of vertex u at layer l or epoch t
$\rho^{(l)}(\{h_u^{(l-1)} : u \in N(v)\})$	$\sum_{w \in N(v)} M_t(h_v^{t-1}, h_w^{t-1}, e_{vw})$	Aggregation at a layer or epoch with $M_t(\cdot)$ and $\rho^{(l)}(\cdot)$ as aggregation functions
$a_v^{(l)}$	m_v^t	Aggregated feature vector
$\phi^{(l)}(\{a_v^{(l)}\})$	$h_v^{t+1} = U_t(h_v^t, m_v^{t+1})$	Combination with functions $U_t(\cdot)$ and $\phi^{(l)}(\cdot)$ in a given layer or epoch

inference. Training is performed via modifications of traditional backpropagation algorithms, which take into account the unique architecture of a GNN. Since a GNN unfolds into L layers similarly to a RNN, most GNNs employ Back-Propagation-Through-Time (BPTT) schemes or variants of it. A popular variant of the BPTT algorithm is the Pineda-Almeida algorithm [54], [55], which relaxes the memory requirements as already mentioned in the seminal work by Scarselli *et al.* [16].

Specifically, in BPTT, firstly a forward pass is performed for the unfolded version of the GNN with its L layers. The loss function ε is then computed and the necessary gradient is backpropagated across the layers. Since the weights are shared across all L layers, they are updated accordingly. This process is carried out recurrently with multiple samples, often grouped in batches, until some user defined target accuracy is reached.

To exemplify the computation occurring during training, bottom charts of Figure 4 represent backpropagation in a two-layer GNN. Again, we show the operations from the perspective of node 1, although all nodes would be realizing similar computations at the same time. The loss function is backpropagated by calculating the gradient of the error with respect to the weights first, via partial derivative over $W^{(2)}$, and then with respect to each vertex's feature vector. The operation is then repeated for the first layer, via its own weight matrix $W^{(1)}$ and each vertex's feature vector. The derivatives of the loss function are, eventually, used to update the weight matrices.

The computation of the loss function depends on the type of learning, which in turn depends on the application at hand. While graph-centric approaches tend to be trained using supervised learning, node-centric approaches are usually trained by means of semi-supervised learning, wherein information of the node features from a portion of the graph, and not the whole graph, is utilized. An example of the former method can be learning if whether a specific new molecule (graph) has a certain property, using a GNN trained with molecules (graphs) whose properties are known beforehand and used as ground truth [12]. For the latter method, an example can be a recommender system. In such a system, a graph represents a store with nodes being shopping items and their features, and edges being relations among items. The output feature vector could describe how likely a given user will be satisfied with a particular item. In this case, a priori complete information is not available and semi-supervised learning from past purchases by this and other users (corresponding to a part of the graph only) is used instead [56].

III. The Evolution of the GNN Field

In this section, we present the evolution of the body of knowledge in the area of GNNs from three different points of view using a Knowledge Graph (KG) representation. We first describe our methodology in Sec. III-A, to then classify an exhaustive set of GNN works by topic in Sec. III-B, by date of publication in Sec. III-C, and by type of algorithm in Sec. III-D. We demonstrate that, as compared to the rest of GNN disciplines, GNN computing is in a very early stage.

A. Methodology

A research field can be understood as a confluence of multiple interrelated works and, hence, a KG is a natural way to illustrate it. In the case of GNNs, taking inspiration on its graph-based structure, using a KG seems especially appropriate. This will allow us to identify densely connected or sparse sub-fields, cross-fertilization, and emerging areas, and also evaluate the evolution of the field over time.

To generate the KG, we created a repository of annotated papers classified by their year of publication. The papers are hand-tagged among a set of possible categories using the title and keywords as main reference. Each paper is assigned a single tag. Further, the references of each paper are extracted by means of the CERMINE library [57]. The generated database is introduced into the Neo4j graph tool [58], which is later used to obtain the different graph-based perspectives of the GNN state of the art.

The generated KG follows a similar structure than other related datasets, where nodes represent papers and edges among nodes represent citation relations among papers. Additionally, the size of each vertex is derived using the page-rank algorithm, which in our case is equivalent to the number of citations that each paper receives. This consequently highlights the significance of each paper within the state of the art. Finally, nodes are color-coded depending on the paper category among the disciplines listed next.

B. A Classification Perspective

Our first treatment of the GNN literature consists in classifying the papers by discipline. Concretely, we define the following taxonomy with topics ranging from formal mathematical aspects, to the algorithms, applications, and computing aspects: *GNN modeling*, *GNN applications*, *GNN complexity*, *GNN algorithms*, *GNN accelerators*, *GNN HW/SW requirements*, and *GNN dataflow*. The description of each topic, together with a discussion of its first works and the list of its references is given in Table V. We also show the percentage of papers that pertain to a given category.

An important finding from our analysis is that the percentage of papers being categorized for *GNN accelerators*, *GNN HW/SW requirements*, and *GNN dataflow* are 10.11%, 7.86% and 3.37%, respectively. These categories mostly relate to the computing side of GNNs as they concern the analysis of computational requirements of GNNs, optimization of GNNs via software, and development of hardware accelerators. We thus observe that a very small percentage of the existing research has approached GNNs from the perspective of computing. We further note that the first works to deal with these topics date back from 2017, when the very first specific paper on GNN acceleration was published. It can be therefore concluded that GNN processing is in its nascent stages of development. This is the main reason for computing aspects not being analyzed in depth in recent GNN surveys [6], [9], [10], [24]–[27], which we aim to address in this work, and also represents an opportunity to make an early impact in the GNN research field.

A second order analysis stems from the careful observation of the KG, which we show in Fig. 5. We have two sub-figures there, the first one representing the aggregation of papers by category and the second one illustrating individual papers independently, yet still clustered by category. In the former case, the size of the node represents the number of papers in a category, whereas the thickness of the edge between two nodes illustrates the relative amount of citations between the papers of a given pair of categories. In the latter case, we can also analyze the connections between the papers within the same category. Several observations can be made from this figure:

(i) The categories related to computing are small yet well-connected to the theoretical side of GNNs, corroborating our earlier observation from Table V.
(ii) The algorithms sub-field is large as many papers have appeared implementing multiple variants in the heterogeneous group of methods that GNN is. We review the evolution of GNN algoritms later in Sec. III-D.
(iii) The applications sub-field is large but sparsely connected internally, which means that application papers are generally not aware of other applications, unless reusing some specific common mechanism. This may be due to the wide variety of application fields for GNNs, ranging from social networks to chemistry, computer networks, or even material science as analyzed in previous sections.
(iv) Algorithms and applications have a strong interconnectivity with each other, as each application paper shall at least mention the algorithms used to implement the proposed system.
(v) The connection from application papers to computing papers is weak. This disconnection may be due to the relative immaturity of the GNN computing field and this may change in upcoming years, especially if applications clearly benefiting from specialized accelerators arise (akin to the appearance of CNN accelerators for computer vision).

C. A Time Perspective

To further understand the state of things in GNNs, we delve deeper into the analysis and visualize the evolution of the field over time. Specifically, we plot the growth of the KG over the years in Fig. 6 and of the amount of published papers in Fig. 7. First works started to appear as soon as 2005 [15] and, at that point, most research efforts were centered around new algorithms and possible applications. Evolution was rather slow for a decade, which we attribute to the lack of a killer application and the modest popularity of deep learning methods at that time. The field exploded around 2016, when CNNs and RNNs were already well established. Such a dramatic growth coincides with the introduction of

TABLE V: The different categories for the classification of the state of the art in GNNs.

Tag Name	Meaning	Origins	References	Percentage
GNN modeling	This category includes the papers that encompass the topics of design and mathematical formulation of GNNs. Other salient design formalisms related to GNNs have been also categorized in this tag.	**2005.** While the most important paper in GNN modeling is from Scarselli *et al.* in 2009, it extends a seminal work from 2005. It defines the mathematical foundation of these GNNs, and thus becomes a fundamental paper in this category.	[15], [16], [45], [59]–[67]	13.48%
GNN applications	Papers with this tag elaborate upon the various applications of GNNs, regardless of the field.	**2005.** Given the ubiquity of graphs in real-world data, this is one of the first sub-fields to have emerged. In their seminal work, Scarselli *et al.* presented the first possible applications together with the first GNN model [68]. Since then, many other applications have appeared.	[11], [17], [18], [68]–[86]	24.72%
GNN complexity	This tag encompasses the papers that explore the complexity within the GNN structure and its operations.	**2009.** The exploration of the complexity of GNN execution may have started with [87] in 2009, which analyzed the complexity for the most common GNNs at that moment. After this, we have to wait until 2017 to find more works that take into account complexity, as datasets become more resource demanding and large-scale applications become apparent.	[87]–[99]	14.60%
GNN algorithms	This tag refers to papers that introduce new algorithm variants to the GNN family, including aspects such as attention, isomorphism, sampling, or new operations at the aggregate–combine phases.	**2009.** We consider [16] as the first unification of multiple similar prior approaches. Others have attempted to do similar generalizations, such as the MPNN from Gilmer *et al.* [12] or the GN from Battaglia *et al.* [6].	[6], [12], [22]–[24], [27], [53], [100]–[115]	25.86%
GNN accelerators	Under this tag, we gather papers that target the acceleration of GNNs either via software or hardware.	**2017.** The earliest paper to tackle the problem of GNN acceleration is [45], in 2017, through a simplification of the algorithm via sampling. More recent works on software in CPUs and GPUs, and hardware acceleration in custom architectures, have also been considered.	[39]–[43], [116]–[120]	10.11%
GNN HW/SW requirements	This tag gathers works that, with the increasing popularity of GNNs as well as the complexity of the data-sets, analyzed the actual computational needs required to address these challenges.	**2018.** This specific sub-field started to gain traction in 2018, with the very first work being [121] where the hardware and software efficiencies in executing GNNs were studied.	[37], [38], [50], [51], [122]–[124]	7.86%
GNN dataflow	Dataflow refers to the movement of data within the processing engine, which becomes crucial for the design of custom accelerators. Hence, under this tag we categorize the papers that formally describe possible dataflow solutions.	**2018.** Two primary works, i.e., [121], which covers scalability in the training, and [46] which covers efficiency for partitioning of the graph data, emerged.	[46], [49], [121]	3.37%

the Graph Convolutional Networks (GCN) [104], one of the first and most popular models for GNNs, later followed by the introduction of the message passing notation and quantum chemistry application in [12].

We further observe that research on GNN computing started in 2017 and, since then, attained a similar growth than the whole field. This trend may be an indicator a strong increase of related works in the near future. Hence, it can be concluded that the area of GNN accelerator design and development is emerging to be a hot topic and, thus, necessitates deeper insights that we provide in upcoming sections.

D. An Algorithm Perspective

GNNs are a family of models with a wide set of possible configurations and design decisions that allow to modulate the inductive bias of the algorithm. Due to their flexibility and potential applicability, we have shown in prior sections how recent years have seen a rapid progress in the development of a myriad of different GNN algorithms. Given their importance, and to guide later analysis of GNN computing aspects, here we briefly summarize the progress in this sub-field. Note that a deep review of existing GNN algorithms is not the main focus of this work. For such an analysis, we refer the reader to more specific surveys [6], [10], [24], [25].

Fig. 8 attempts to provide a classification of the different GNN algorithms that can be found in the literature. The taxonomy does not aim to be exhaustive, but rather provide a broad overview and guide the reader through the history of GNN algorithms.

Pre-GNN techniques. The main appeal of GNNs is to extract the relevant relational information from a graph using ML methods. Prior to the advent of GNNs, however, the problem of relational information extraction from graphs was performed via other techniques [9], [28]. The main approach was to pre-process the graph to condense the information in a low-dimensional space, i.e. a graph embedding, and thus make it amenable to traditional ML algorithms. A first example of such a historical approach are the circular fingerprints [125], which summarize molecular structure graphs through atom neighborhoods.

Amongst the modern day approaches, Graph Kernels (GK) are a family of kernel methods capable of extracting graph-level embeddings to perform classification tasks. In order to operate, they compute the similarity between pairs of graphs, using the so-called kernel functions. These functions, which must satisfy positive definiteness, transform the input graph to a higher dimensional feature space where it is easier to perform the aforesaid comparisons. Multiple GK approaches have been proposed, many of them yielding promising results [126], [127]. Notably, one of the most relevant approaches is the random walk kernel, wherein random walks are performed on the graphs while simultaneously counting the matching walks

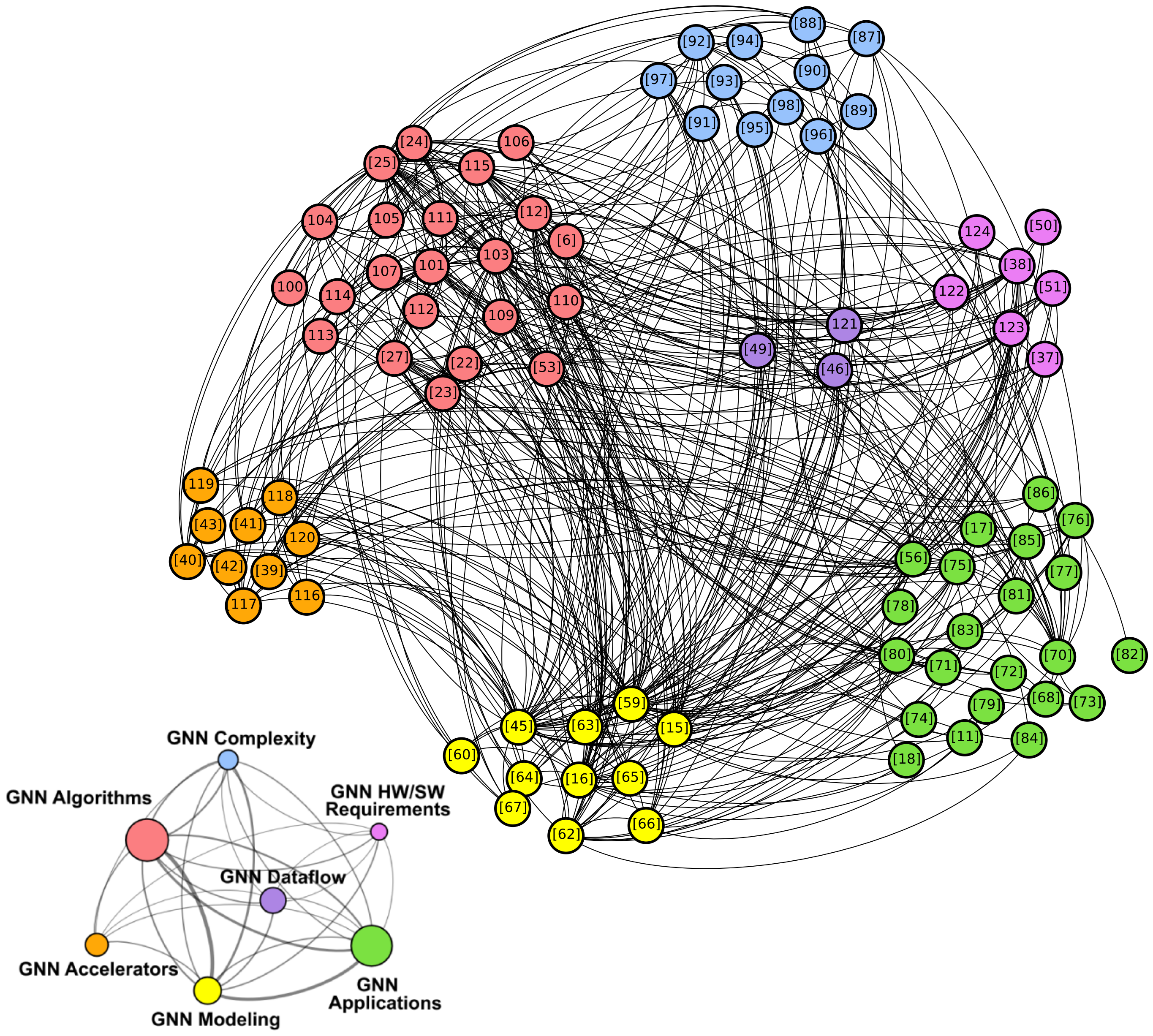

Fig. 5: Full knowledge graph representation as of September 2020.

[128]. Since it is computationally hard to compute all the possible walks, recursive algorithms are utilized in practice. For further literature on GKs, we refer the reader to [129], [130].

As compared to GNNs, GKs are easier to train because they have less hyperparameters, which on the other hand limits their performance. The main reason stems in the loss of potential information incurred by the process of graph embedding. Thus, to achieve acceptable performance, GKs require handcrafted (not learned) feature maps, whilst GNNs do not. GNNs use the inherent graph structure as a powerful and expressive form of defining the neural network, instead of distilling the essence of the graph to feed a conventional neural network.

General GNN classifications. Since the seminal work by Scarselli et al. [16], multiple approaches have been published with the aim of elaborating and complementing the GNN concept [62], [66], [131], [132] and many classifications can be carried out. For instance, there is a distinction between transductive learning and inductive learning applied to GNNs. Transductive learning applied in algorithms such as DeepWalk [133], Node2vec [134] or Graph2vec [135] is tightly coupled with semi-supervised learning, means that inference occurs over nodes of a given graph after training over that same graph only. Thus, new graphs require additional training. On the other hand, inductive learning means that sets of graphs are provided to derive general rules, which they can be subsequently applied to generalize about unseen graphs. For instance, Chen *et al.* [96] and GraphSAGE [45] are examples of inductive GNNs.

Another common distinction relates to the fundamental model upon which the GNN is built, and mostly differentiates

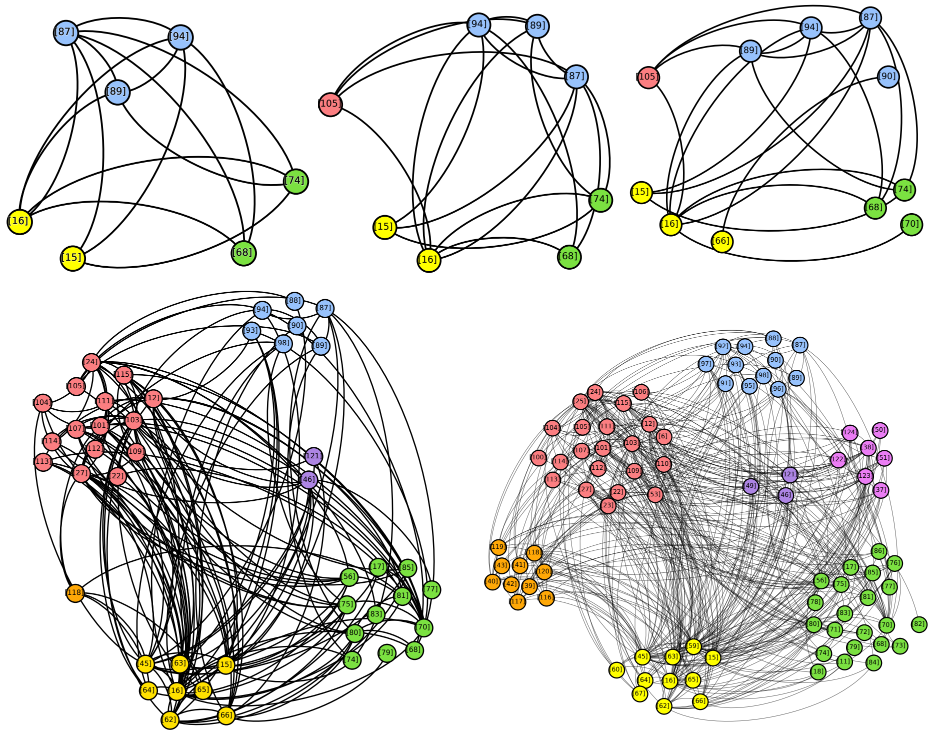

Fig. 6: Evolution of the GNN knowledge graph over the years 2009, 2012, 2015, 2018, and 2020. The color code is consistent with that of Figure 5.

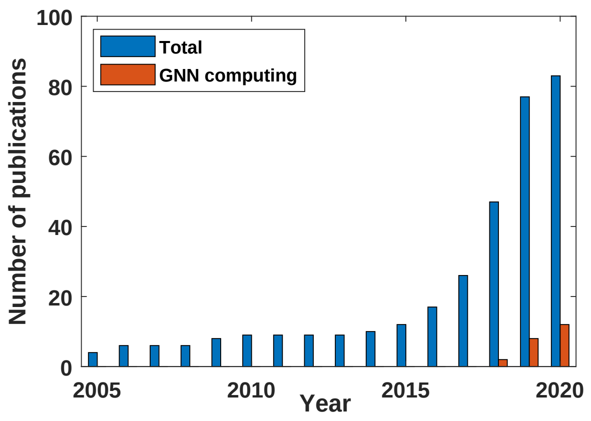

Fig. 7: Cumulative number of papers published over the years in GNNs in general and in GNN computing in particular.

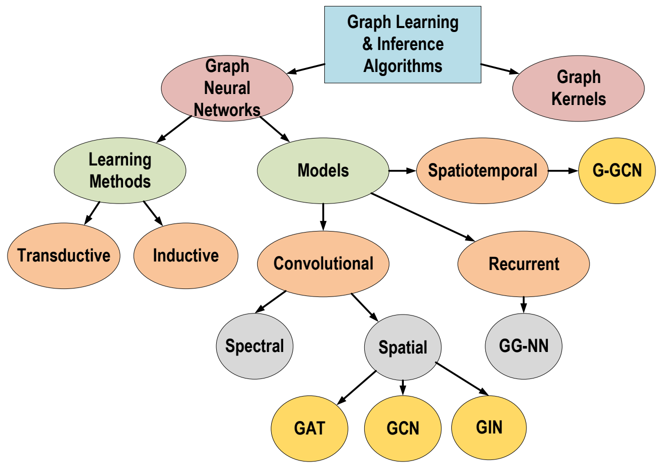

Fig. 8: A classification tree for GNN algorithms.

between recurrent-based GNNs, convolutional-based GNNs, and spatiotemporal GNNs [25]. On the one hand, recurrent-based GNNs refer to the initial GNN models including that of Scarselli [16], which follow a the general iterative scheme described in previous sections with recurrent units in the update. Other examples of recurrent-based GNNs are CommNet [136], which operates over simple aggregations without edge transformations, or Gated Graph Neural Networks (GG-NN, [103]), which use gated recurrent units [137] as the update function to improve convergence. On the other hand, convolutional-based GNNs essentially expand the idea of convolution in the graph space by applying weights and non-linear activations to the aggregation of multiple neighbours data, with many variants [63], [101], [138]–[146] as we explain later in this section. Finally, spatiotemporal GNNs use both the spatial approach of the convolutions with the temporal approach of the recurrent units. An example is the network in gated graph convolutional network (G-GCN) from [147].

Another branch of GNNs are the so-called Graph Autoencoders (GAE) [104]. These GNNs are generative, which means that they convert a graph into a latent representation that can be later expanded to generate to a new graph close in structure to the original one. The first step, which is named encoding, creates graph and node embeddings with essential structural information and may employ GCNs to that end, or not [148] presents a simpler approach. The second step, which is decoding, generates graph representations from the low-dimension vector containing the encoded structural features. Kullback-Liebler divergence can then be utilized as the loss function to determine the differences in the distributions for a pool of graphs. Earlier types of graph generative algorithms, targeting a modeling of a particular graph family [149], included certain assumptions on the distributions [150], used generative adversarial networks [151], or autoregressive models [152].

Finally, within the body of knowledge on convolutional GNNs, one can be further divide them into spectral-based GNNs [66] and spatial-based GNNs [79]. The former refers to models based on spectral graph theory, in which by following graph signal processing techniques and by means of eigenvalue decomposition, filters can be introduced. However, they are computationally expensive methods, since the entire graph must be considered at once. On the other hand, spatial-based GNNs are much more computationally affordable, since they only need to perform convolutions using neighbor features and local processing [79]. Notably, spectral and spatial domains are generalized in [153], but ultimately the latter outweighs the former and as a consequence, most of the convolutional GNNs today are spatial-based. In addition to the computational complexity, another reason for the dominance of spatial-based GNNs is their flexibility and scalability, this is, spatial-based GNNs are known to perform better in unseen graph shapes and to work better with huge graphs, which is crucial for application areas such as social networks.

Spatial-based convolutional GNNs: a representative family. Within spatial-based convolutional GNN, the work in [59] stands out by being one of the pioneers of GCN. They take into account the degree of neighboring nodes when weighing the aggregation stage (c.f. Section II). It is worth noting that GCN is one of the most fast-evolving areas under the GNNs umbrella [63], [101], [138]–[146]. For instance, after the initial GCN, GraphSAGE incorporated information of self-node features from previous layers in the update function, and presented new aggregation methods, such as LSTM, which further improved upon the performance of their baselines [45]. GraphSAGE also introduces a sampling operation, by which mini-batches are considered instead of the full graph, to help controlling the required memory and runtime. FastGCN [101] further uses the sampling idea and integrates other strategies to speed up computations, such as evaluating integral formulations using Monte Carlo sampling. SGC [139] removes some complexity in GCNs by removing nonlinearities between layers and obtaining a single linear transformation. R-GCN [63] uses modified GCNs to model highly multi-relational data in knowledge graphs. Another simplifying operation is the differential pooling of *DiffPool* [154], which puts nodes together into clusters hierarchically so that later layers operate on coarser graphs.

Graph Attention Networks (GAT) opens a new cluster of GCN works. They update the graph node features through a pairwise function between the nodes, and considering it as a weight. That is, the importance of a node to other nodes will also depend on the features of the connection between the nodes, and hence, it can be learnt as well. This is formally referred to as an anisotropic update [67]. Neighbouring nodes can subsequently share their information by means of a learnt function. This allows to operate with a learnt attention mechanism. Specifically, the utility of the edges is learnt as well. In order to further emphasize on the potential of the GATs, the Natural Language Processing (NLP) paradigm can be understood as a multi-head GAT [155].

Finally, Graph Isomorphism Network (GIN) [53] are also a popular algorithm developed upon the idea of leveraging Weisfeiler & Lehman (WL) isomorphism test [156] to improve expressive power of GNNs, this is, to making GNNs more sensitive to the different input graphs. This is done by introducing a learnable parameter that modulates the weights of central nodes, which allows GIN to better distinguish among graphs simply by the embedding they generate.

Comprehensive frameworks. One final aspect worth mentioning is that, within this multitude of algorithms, several groups have attempted to unify methods. One of the most popular ones is the message passing scheme [12], [157], whose operation and description are amenable to convolutional networks for learning molecular fingerprints [70], the classification methodology with GCN from [59], the interactive networks utilized for learning relationships and features [81], or also different flavours of Gated GNNs, to name a few. A further approach is that of the Non-Local Neural Networks (NLNN) [22] aimed at unifying various attention approaches including GATs. These generally do not include edges features or aggregations and, instead, just involve pairwise scalar attention weights between nodes. Finally, both MPNN and NLNN are included into a further approach to unification referred to as graph networks and proposed in [6]. There, update functions applied to nodes, edges, or the complete graph are treated as differentiated blocks. The combination or repetition of several of these blocks gives rise to the different types of GNN found in the literature.

From the perspective of computation, several programming abstractions are considered to support all possible operations within any GNN, generally compatible with the aggregate-update model. Among them, we highlight the *Scatter-ApplyEdge-Gather-ApplyVertex with Neural Networks* (SAGA-NN) from [38] or the *Gather-Reduce-Transform-Activate* (GReTA) from [49]. These are described in more detail in the next section.

IV. The Revolution of GNN Acceleration

The optimization of ML algorithms and the building of custom hardware for high performance and energy efficiency has experienced an unprecedented growth in the recent years [2], [36]. This has come shortly after academia and industry have unveiled the outstanding potential of DNN algorithms and their all-pervasive applicability.

As evidenced in previous sections, the field of GNNs is arriving at a similar turning point. At the time of this writing,

TABLE VI: Operations in popular GNN algorithms.

Algorithm	**Aggregation** (a)	**Combination** (h^{l+1})
GCN [59]	$mean(N(h^l))$	$ReLU(W_l \cdot a)$
GIN [53]	$mean(N(h^l))$	$MLP(W \cdot ((1+\epsilon^l) \cdot h^l + a)$
GS-mean [45]	$mean(N(h^l))$	$\sigma(W_l \cdot Concat(a, h^l))$
GS-max [45]	$max_{j \in N(h^l)} \sigma(W_l^1 \cdot h_j^l)$	$\sigma(W_l^2 \cdot Concat(a, h^l))$
GS-LSTM [45]	$LSTM(rand(N(h^l)))$	$\sigma(W_l \cdot Concat(a, h^l))$
GAT [115]	$\sum_{j \in N(h^l)} \alpha_j W h_j^l$	$\sigma(a)$
HighwayGCN [158]	$\sigma(W^l \cdot h^l + b^l)$	$h^{l+1} \odot a + h^l \odot (1-a)$
GRN [40]	$mean(N(h^l))$	$GRU(h^l, W^l \cdot a)$

Notation: σ is a nonlinear function, α_j is the attention coefficient, b is the bias, $\odot$ is a dot-product, $Concat$ is matrix concatenation, MLP is a multi-layer perceptron, GRU a gated recurrent unit, and $LSTM$ is Long short-term memory.

research in GNN methods is already extensive and keeps refining the algorithms and investigating new applications with high potential impact. Therefore, a key research aspect in the years ahead will be how to compute GNNs efficiently to realize their full potential.

GNN computing presents a set of unique challenges [50], [159] that have rendered existing libraries and hardware platforms inefficient, including:

(i) The existence of multiple GNN variants, inductive and transductive, with a variety of aggregation and combination functions as illustrated in Table VI.
(ii) The dependence of computation on the characteristics of the input graph in terms size, sparsity, clustering, or the length of the associated feature vectors. Graph connectivity may follow a power-law distribution, be evenly distributed, or be bipartite.
(iii) A unique combination of computing characteristics from deep learning and graph processing, leading to alternate execution patterns. The former is computation-bound and involves dense matrix multiplications [160], whereas the latter is memory-bound and involves sparse algebra [161].
(iv) The need to scale to very large graphs of upto billions of edges in certain applications.

In light of these challenges, GNNs call for new solutions both in software and hardware. On the software side, several libraries have been proposed to improve the support for GNNs and handle its multiple variants, being the extensions of popular libraries such as PyTorch or Tensorflow (TF) [47], [162], [163] clear examples. On the hardware side, new accelerator architectures have been surfacing recently [40], [42], [117], [164] that attempt to deal with the flexibility and scalability challenges of GNNs. In the next subsections, we provide an exhaustive overview of existing techniques.

A. Software Frameworks and Accelerators

The challenges of GNN processing rendered traditional DNN libraries inefficient. To bridge this gap, very recent works have started investigating how to adapt the libraries to (i) provide easy to program interfaces to implement multiple GNN variants, (ii) handle the sparsity of GNN operations efficiently in widespread GPU hardware, (iii) scale computations to large-scale graphs and multiple GPUs.

In the following, we review a comprehensive selection of software frameworks and accelerators, listed in Table VII. The analysis does not include GunRock [165] or GE-SpMM [166] for different reasons. GunRock, despite implementing GraphSAGE in its latest versions, it is originally a graph processing library. GE-SpMM, although claiming to be tailored to GNNs, is an acceleration method for general-purpose sparse matrix multiplication in GPUs.

A first observation from Table VII is that software frameworks have been tested for a wide variety of GNN algorithms and relevant datasets. Around 20 different GNN variants have been evaluated, being GCN, GS, and GIN the most common. Even though Amazon, Reddit, Protein, Cora, or CiteSeer datasets are popular in the community, a lack of a widely adopted benchmark suite [167] makes the datasets to vary widely. It is worth noting, however, that graphs can range from hundreds of edges in chemistry applications to billions of edges in large-scale recommendation systems. As we see next, performance comparisons are scarce, but generally take PyG, TF, and DGL as baselines.

PyTorch Geometric (PyG). PyG [47] is a widespread library that is built upon PyTorch and that provides support for relational learning, illustrated in a myriad of algorithms[1]. The key aspect is the definition of a message passing interface with definition of `message` and `update` functions for neighbourhood aggregation and combination, respectively, and multiple pooling operations. To accelerate GNN processing, PyG handles sparsity via dedicated GPU scatter and gather kernels that operate in all edges and nodes in parallel, instead of using sparse matrix multiplication kernels. Relevantly, Facebook released Pytorch-BigGraph [171], a library that allows to process arbitrarily large graphs by introducing partitioning and distributed processing and that could complement PyG.

Deep Graph Library (DGL). DGL [48] is a recent library that works on top of TF, PyTorch, or MXNet, and provides plenty of examples and code for multiple GNNs[2]. The library defines three functions: `message` for edge aggregation and update and `reduce` and `update` for aggregation and combination at the nodes. To boost performance, DGL takes a matrix multiplication approach and leverages specialized kernels for GPUs or TPUs. In particular, both sampled dense-dense and sparse matrix multiplications are considered together with node, edge or feature parallelization. As discussed in their work [48], DGL uses heuristics to choose among the different options as the optimal parallelization scheme depends on multiple factors including the input graph. Thanks to this approach, DGL claims to achieve an order of magnitude faster training than PyG.

NeuGraph. Microsoft Research led one of the first specialized frameworks for parallel processing of GNNs in GPUs, NeuGraph [38]. Although it is built on top of TF, NeuGraph is not open source at the time of this writing. The framework implements a programming model, SAGA-NN, based on the functions `Scatter` for edge aggregation, `ApplyEdge` for edge combination, `Gather` for node aggregation, and `ApplyVertex` for node combination. Scatter-gather kernels are used in the functions of the same name, whereas matrix

[1]Examples at https://github.com/rusty1s/pytorch_geometric
[2]Code at https://www.dgl.ai/

TABLE VII: State of the art in software frameworks and accelerators for GNNs (GS = GraphSAGE)

Name	Main Features	Evaluation		
		Algorithms	Datasets	Baselines
PyG [47]	• Leverages widespread adoption of PyTorch. • Wide variety of example codes available. • Use of scatter-gather kernels + node/edge parallelism. Evaluated in GPU.	GCN, GAT, SGC, GS, GIN, etc...	Cora, CiteSeer, PubMed, MUTAG, Proteins, Collab, IMDB, Reddit	DGL
DGL [48]	• Library compatible with TF, PyTorch and MXNet. • Deep documentation and support, tutorials. • Based on matrix-mul kernels. Evaluation in CPU and GPU.	GCN, GAT, SGC, GS, GIN, R-GCN, GCMC	Reddit, OGB (Arxiv, Protein, Product, Citation, PPA), Movielens	PyG
NeuGraph [38]	• Implementation and evaluation targeting scaling to multiple GPUs. • Four-function model allowing for updates at both edges and nodes. • Deeply optimized partitioning, scheduling, pipelining, GPU transfers. • Built on TF, not open sourced.	GCN, CommNet, GG-NN	pubmed, blog, reddit, enwiki, amazon	DGL, TF
AliGraph (Euler?) [118]	• Implementation targeting large-scale graphs and distributed systems. • Emphasis on distributed storage and partitoning (four algorithms) • Only work evaluating GNNs for heterogeneous and dynamic graphs, and huge datsets (up to 483M edges, 6.5B edges). Built on top of TF.	GS, six in-house algorithms	Amazon, Taobao	N/A
AGL [168]	• Aiming for scalability, fault tolerance, and integrality. • The system distributes the workload with well-known MapReduce.	GCN, GS, GAT	Cora, PPI, UUG,	PyG, DGL
ROC [124]	• Implemented on top of FlexFlow [169]. • Key optimizations: dynamic partitioning and memory management. • Evaluation with single and multiple GPUs.	GCN, CommNet, GIN, GS, FastGCN	Pubmed, PPI, Reddit, Amazon	TF, DGL, PyG, NeuGraph
GNN Advisor [39]	• Unique runtime profiling of graph information (degree, feature size, communities) to guide GPU processing • Extensive comparison with similar frameworks in single GPU.	GCN, GIN	CiteSeer, Cora, Pubmed, PPI; Proteins, Yeast, DD, Twitter, SW-620H; amazon, artist, blog	DGL, PyG, NeuGraph, GunRock
Tripathy *et al.* [37]	• Not a framework, but a comparison between parallelization algorithms for large-scale graphs (up to 14M and 1.3B edges in the evaluation). • Tested in a supercomputer, with multi-GPU nodes	GCN	Reddit, Amazon, Protein	N/A
PCGCN [123]	• Motivated by power-law distribution of node degrees. • Optimized partitioning to generate dense matrices. • Dual execution mode depending on sparsity of each partition. • Built on top of TF, evaluated in single GPU.	GCN	pubmed, blog, youtube, C1000-9, MANN-a81, reddit, synthetic graphs (RMAT)	TF, DGL, PyG
HAG [51]	• Removes redundant sums in aggregation by *fusing* similar nodes. • Runtime algorithm to *fuse* nodes only if predicted beneficial. • The impact on operation reduction is independent of hardware, but the impact on execution speed is not.	GCN, GIN, SGC	BZR, PPI, reddit, IMDB, COLLAB	N/A
FeatGraph [170]	• Optimization of matrix mult. kernels for aggregation and combination. • Allows user to define combination functions and their optimization.	GCN, GS, GAT	OGB (Proteins), Reedit, sythetic graphs	GunRock
GReTA [49]	• Programming abstraction with user-defined functions similar to SAGA (NeuGraph), targeting accelerators and applicability to any GNN variant. • Evaluation based on GRIP (see Table VIII).	GCN, GS, G-GCN, GIN	youtube, livejournal, pokec, reddit	N/A

multiplication primitives are used in the combination functions. NeuGraph also features a number of optimizations to accelerate GNN computing. First, the partitioning of large graphs performed via the Kernighan-Lin algorithm to make partitions denser and minimize the transfers between partitions, which harm performance. Second, scheduling of partitions to the GPU is optimized by batching together small sparse partitions that can be computed together [172], and also profiling transfer and computation times in first GNN layer to later pipeline different chunks perfectly. Third, NeuGraph also eliminates redundant computation by fusing multiple edges together. Finally, it allow to scale GNN to multiple GPUs by distributing the computation, and optimizes the transfer of information by using a ring-based dataflow that minimizes contention at the interconnect.

AliGraph/Euler. Developed by the AliBaba group and open-sourced with the name of Euler, AliGraph is a GNN framework built on top of TF [118]. The framework is thought for the processing of very large and dynamic graphs in large-scale computing systems, and is currently used in recommendation services at AliBaba. It implements three layers, namely: *storage*, that implements partitioning with four different algorithms, but in this case to store the graph in a distributed way; *sampling*, which unlike other frameworks, allows to define custom sampling of a nodes' neighbourhood relevant to algorithms such as GraphSAGE; and *operator*, which implements the aggregation and combination functions. In overall, the AliGraph is unique due to its distributed approach and the many optimizations made at the storage layer to minimize data movement, such as the use of four different partitioning algorithms depending on the characteristics of the graph, or caching important vertices in multiple machines to reduce long misses.

AGL. AGL [168] is a framework created specifically for industrial deployments of massive GNNs. To that end, the authors emphasize their scalability, fault tolerance, and use of existing widespread methods for distributing the computation. In particular, AGL uses MapReduce [173] to that end and tests the proposed system in CPU clusters. The framework has three modules, one for creating independent neighbourhoods that can be processed in parallel for training, one for the optimized training, as well as one for the slicing of the graph and calculation of inference. Numerous optimizations are proposed in the sampling and indexing of the graph, partitioning and pruning, and at the pipelining of the training computation.

ROC. ROC [124] is another GNN framework targeting multi-GPU systems, in this case built on top of FlexFlow [169]. Similarly to AliGraph or AGL, ROC is able to distribute large graphs to multiple machines. However, this framework differs from others in that the partitioning method and memory management is performed with dynamic methods providing extra acceleration. First, ROC uses an online linear regression model to approach partitioning optimally. This model uses the training iterations to learn the best strategy of a specific graph, outperforming static methods significantly. Second, memory management is treated as a cost minimization problem and solved via an online algorithm that finds where to best store each partition. The authors demonstrate that such acceleration methods provide better scalability than DGL and PyG in single GPUs, and better scaling to multiple GPUs than NeuGraph.

GNNAdvisor. The work by Wang *et al.* [39] presents a runtime system that aims to systematically accelerate GNNs on GPUs. Instead of treating this problem via abstract models as done in ROC, GNNAdvisor does an online profiling of the input graph and GNN operations to guide the memory and workload management agents at the GPU. In particular, it leverages (i) the node degree to fine-tune the group-based workload management of the GPU, (ii) the size of the node embedding to optimize workload sharing, and (iii) the existing of communities within the graph to guide partitioning and scheduling. While the two first features are trivial to obtain, community detection is generally harder. In this case, the authors use a combination of node renumbering and Reverse CuthillâĂŞMcKee algorithm to reorder the adjacency matrix in a way that dense partitions are available. Thanks to all these techniques, the authors claim 3×-4× speedup over DGL, PyG, and NeuGraph in a high-end GPU.

Tripathy *et al.* In this work, the authors compare multiple parallelization algorithms that partition and distribute the GNN in multiple GPU clusters, i.e., 1D, 1.5D, 2D and 3D algorithms, and model the tradeoff between inter-GPU communication and memory requirements of these setups analytically and for training. The model takes a large adjacency matrix and breaks it down to a fixed amount of processes depending on the algorithm. Then, an analysis is made on the amount of effectual operations and results to be communicated across the GPUs. Their implementation over PyG shows promising scalability and nominates the 1.5-D algorithm as a promising and balanced alternative, although the best algorithm depends on the characteristics of the input graph.

PCGCN. The paper by Tian and co-authors [123] present a partition-centric approach to acceleration of GNNs in GPUs, which they implement on top of TF. The contribution is motivated by the power-law distribution of the node degrees in a graph, which largely affects partitioning. PCGCN applies a locality-aware partitioning, METIS [174], that helps obtaining dense sub-matrices. That, however, does not prevent sparse partitions to appear. To combat this, PCGCN profiles the partitions at runtime and applies a dual-mode of operation: dense matrix representation and multiplication kernels when dense, and column-sparse representation and sparse kernels otherwise. In the paper, the authors compare their implementation with vanilla TF, and also DGL and PyG, and report the lowest speedup across libraries. Even in this case, PCGCN always speeds up execution and achieves upto 8.8× in highly clustered graphs.

HAG. This work presents the concept of *Hierarchically Aggregated computation Graph* (HAG) [51]. The authors make the observation that many of the operations made during the aggregation stage are repeated multiple times when nodes share similar neighbourhoods. In response to this, HAGs are presented as an alternative representation that proactively “fuses" nodes with common neighbourhoods, removing redundant aggregations during the execution of any GNN. Since the search of similarly-connected nodes can be expensive, HAG employs a cost function to estimate the cost of certain node fusions, to then adopt a search algortihm affordable for runtime. With only 0.1% of memory overhead, HAG reduces the amount of aggregations by 6.3×.

FeatGraph. Developed in collaboration with Amazon, FeatGraph [170] proposes to optimize kernels of aggregation (graph traversal) and combination (feature computation) separately. Different from other frameworks, here the user can define the combination function and ways to parallelize it, so that the scheduler can take it into account. As optimizations, FeatGraph also proposes to combine graph partitioning with feature dimension tiling and to adopt a hybrid partitioning scheme for GPUs.

GReTA GReTA [49] is a processing abstraction for GNNs aiming at simplifying their representation for hardware implementations. Rather than boosting performance, GReTA aims to be an interface to hardware accelerators. It consists of four user-defined functions, namely `Gather` and `Reduce` to describe the aggregation, and `Transform` and `Activate` to describe the combination. These user-defined functions enable certain flexibility to accommodate different GNN types. GReTA also discusses partitioning briefly, which is exemplified in a hardware acceleration prototype called GRIP [117], which we describe in the next section.

B. Hardware Accelerators

We have seen that software approaches presented above streamline the execution of GNNs in generic computing platforms present in most computing systems, achieving significant speedups. Fewer works analyzed the performance of

GNN training in a TPU, which is typically used in dense deep learning, but with similar results [122].

However, a pertinent question is whether custom hardware accelerators can tackle the unique challenges of GNN computing and live up to the promise of order-of-magnitude improvements that, to cite an example, have been already achieved in CNNs [36]. Pursuing this goal, several hardware accelerators have emerged which attempt to handle the extreme density and alternating computing requirements of GNNs. We next discuss all the designs published to date, using as reference the schematic diagrams of their architecture shown in Fig. 9. The figure also tries to classify the architectures depending on how tightly coupled they are, from unified to tiled, and how easy is to adapt them to multiple GNN variants.

A summary of the main features of the accelerators and evaluated algorithms and datasets is given in Table VIII. We observe that most works revolve around the GCN algorithm, which is popular and easy to illustrate. Those that evaluate multiple algorithms generally try to demonstrate their generality. Datasets are generally smaller than in software acceleration works, mainly because of the memory limitations of hardware accelerators in inference and the cost of simulating hardware architectures. Cora, CiteSeer, and Redit are the most common ones. While performance comparisons are difficult due to the many variables involved, most works use CPUs and GPUs as baselines and, in some cases, even one hardware accelerator. There is no consensus on which software framework shall be used in the baselines.

EnGN. Among the first accelerators to appear, EnGN [40] presents a unified architecture heavily inspired by CNN accelerators. The GNN is fundamentally treated as concatenated matrix multiplication of feature vectors, adjacency matrices, and weights –all scheduled in a single dataflow. An array of clustered Processing Elements (PEs) is fed by independent banks for the features, edges, and weights to compute the combination function. To perform the aggregation, each column of PEs is interconnected through a ring and results are passed along and added according to the adjacency matrix in a process the authors call Ring-Edge Reduce (RER).

In EnGN, sparsity is handled with several optimizations. First, the RER aggregation may lead to multiple ineffectual computations for sparsely connected nodes. To avoid this, EnGN reorders edges on the fly in each step of the RER. Second, PE clusters are attached to a degree-aware vertex cache that holds data regarding high-degree vertices. The reasoning is that well-connected vertices will appear multiple times during the computation and caching them will provide high benefit at modest cost. Other optimized design decisions relate to the order of the matrix multiplications when the aggregation function is sum, which affects the total number of operations, or the tiling strategy, which affects data reuse and I/O cost.

HyGCN. The authors HyGCN [41] build upon the observation that GNNs present two main alternating phases of opposed computation needs, to introduce a hybrid architecture for GCNs. HyGCN is composed of separate dedicated engines for the aggregation and the combination stages, plus a control mechanism that coordinates the pipelined execution of both functions. Being dense, the combination stage is computed via a conventional systolic array approach. The aggregation stage has a more elaborated architecture featuring a sampler, an edge scheduler, and a sparsity eliminator that feeds a set of SIMD cores.

In HyGCN, sparsity is handled at the aggregation engine thanks to efficient scheduling and the sparsity eliminator. The latter takes a window-based sliding and shrinking approach to dynamically adapt to varying degrees of sparse multiplications. To further adapt to the workloads, HyGCN allows to group the SIMD cores in aggregation and the PEs in combination in different ways depending on the size of feature vectors. Finally, special attention is placed to the design of the inter-engine coordinator to optimize memory accesses and allow fine-grained pipelining of the execution towards maximizing parallelism dynamically.

AWB-GCN. The *Autotuning-Workload-Balancing* GCN accelerator [42] mainly advocates for an aggressive adaptation to the structural sparsity of the GNN. The authors motivate their design by analyzing the power-law distribution of most graphs, arguing that some parts of the computation will be dense and others extraordinarily sparse, creating unbalances. To address the imbalance, the architecture develops a custom matrix multiplication engine with efficient support of skipping zeros. To that end, data from memory is fed via a task distributor and queue (TDQ) to a set of PEs and accumulators. The TDQ takes two designs adapted to when sparsity is moderate or high.

The key of AWB-GCN is that it implements three workload balancing functions. The first is local and tries to balance the load among neighboring PEs. The second is remote and attempts to pour overflowing computation from a busy PE to a single remote underutilized PE. The third one takes the load of extremely busy PEs processing very dense node clusters and divides across multiple idle PEs. To support that, AWB-GCN provisions hardware at the TDQ and the connections to the PEs to allow the remapping of nodes to remote PEs and to take them back for coherent aggregation. Moreover, all decisions about balanced are taken based on information extracted from simple counting at the queues.

GRIP. A key aspect of most existing accelerators is that they focus on GCNs as a relevant GNN algorithm. In contrast, the GRIP accelerator [117] leverages the abstraction of GReTA [49] to develop a general accelerator for any GNN variant, allowing to perform edge and node updates with user-defined functions. The GRIP architecture reflects this by having separated and custom units and accumulators for both edges (gather, reduce) and vertices (transform, activate). A control unit orchestrates data movement between the different units and respective buffers. In the sample implementation, GRIP divides the edge update unit into lanes to execute vertices simultaneously and takes an input-stationary dataflow for the vertex update unit. Among the optimizations made, we found pipelining and tiling adapted to the particularities of the implemented dataflows, similar to that of other accelerators.

Auten *et al.* Unlike most other accelerators, this work [43]

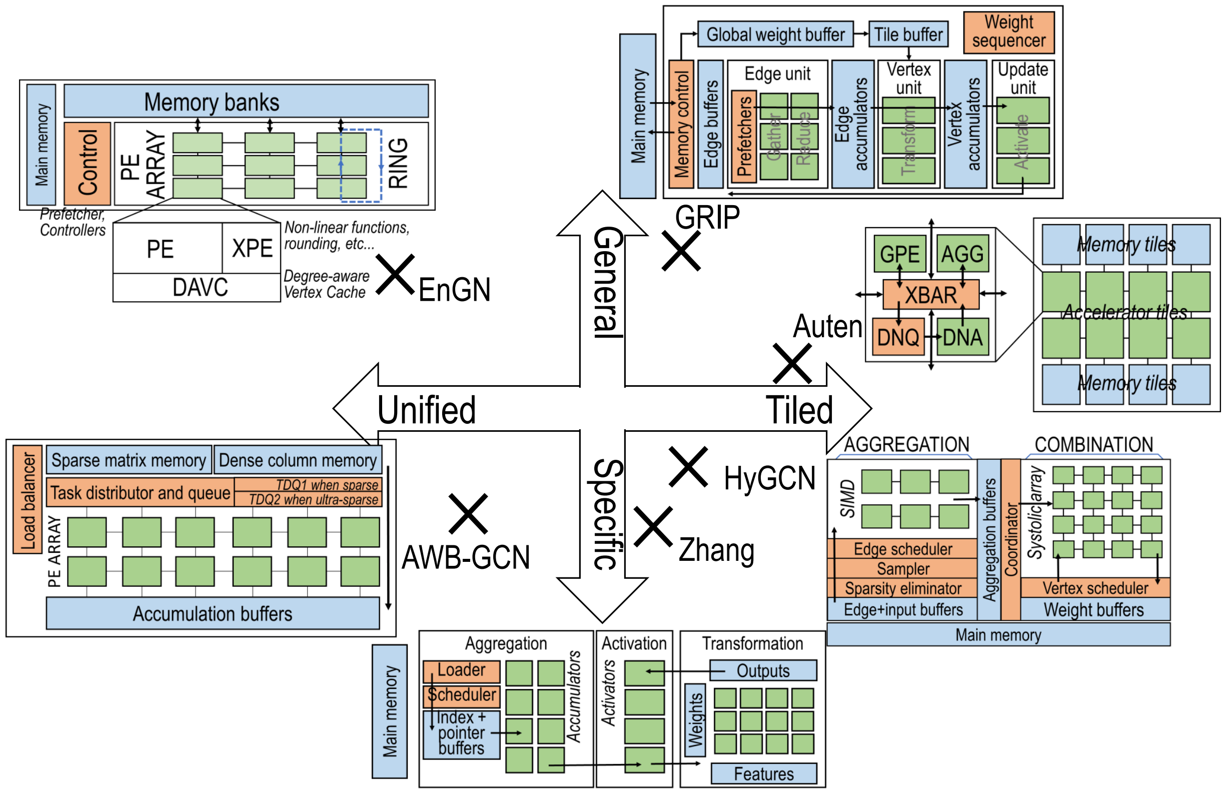

Fig. 9: Classification and schematic representation of hardware accelerators for GNN inference. Green, blue, and red squares represent processors, memory, and control units, respectively.

TABLE VIII: State of the art in hardware accelerators for GNNs.

Name	Main Features	Evaluation		
		Algorithms	Datasets	Baselines
EnGN [40]	• Unified architecture with dense hardware, single dataflow, generalizable to many GNN variants. • Aggregation via Ring-Edge Reduction (RER). • Optimizations: edge reordering, degree-aware vertex cache, scheduling.	GCN, GS, GG-NN, GRN, R-GCN	Cora, PubMed, Nell, Reddit, Enwiki, Amazon, Synthetic (RMAT), AIFB, MUTAG, BGS, AM	CPU-DGL, GPU-DGL, CPU-PyG, GPU-PyG, HyGCN
HyGCN [41]	• Hybrid architecture with specialized aggregate and combine blocks. • Fine-grained pipelining across blocks via inter-phase coordinator. • Sparsity eliminator based on window sliding and shrinking approach. • Focused on GCNs, unclear how to generalize (missing edge updates).	GCN, GSC, GIN, DiffPool	IMDB, Cora, CiteSeer, COLLAB, PubMed, Reddit	CPU-PyG, GPU-PyG
AWB-GCN [42]	• Aggressive adaptation (balancing) to varying GNN workloads via three different load sharing techniques (local + remote) at runtime. • Controller with distinct operation depending of sparsity of each partition. • Focused on GCNs, unclear how to generalize.	GCN	Cora, CiteSeer, PubMed, Reddit, Nell	CPU-PyG, GPU-PyG, FPGA, HyGCN
GRIP [117]	• Uses the GReTA abstraction [49], allegedly generalizable to any GNN. • Actual implementation with techniques similar to HyGCN.	GCN, G-GCN, GS, GIN	Youtube, Livejournal, Pokec, Reddit	CPU-TF, GPU-TF
Auten *et al.* [43]	• Tiled architecture, ready for scale-out via Network-on-Chip. • Techniques similar to HyGCN, less specialized but easier to generalize.	GCN, GAT, MPNN, PGNN	Cora, CiteSeer, PubMed, QM9_1000, DBLP_1	CPU, GPU
Zhang *et al.* [116]	• Combination of offline software acceleration (redundancy elimination + node reordering) and hardware acceleration in FPGA. • Optimizations: double buffering technique to hide latency of additions, node+feature parallelism, dual pipelining mode depending of order of matrix multiplications	GCN	Flickr, Reddit, Yelp	CPU-TF, GPU-TF, CPU-C++, GPU-C++
GraphACT [119]	• Only accelerator evaluating training and modeling memory footprint. • CPU+FPGA platform. Optimizations rely on judicious load balancing, scheduling, batching, removal of redundant aggregation operations.	GCN	PPI, Reddit, Yelp	CPU, GPU

proposes a modular architecture for convolutional GNNs. The basic unit of the accelerator is a tile composed by an aggregator module (AGG), a DNN accelerator module (DNA), a DNN queue (DNQ) and a graph PE (GPE), all of them connected to an on-chip router. Thus, the architecture can be scaled out by interconnecting multiple tiles among them and with memory. Within each tile, the architecture has a similar structure than HyGCN, with the DNA being an array for dense multiplication, the AGG an edge-controlled adder, the DNQ taking the role of inter-engine buffer, and the GPE controlling execution. In this case, however, the GPE is a lightweight CPU managing multiple threads rather than an optimized controller.

Zhang *et al.* The work by Zhang and co-authors [116] presents a combination of software and hardware acceleration for GCNs. On the one hand, the graph is pre-processed via a redundancy elimination mechanism similar to that of [51] and a node reordering similar to that of [39]. Pre-processing is done offline and is justified for the repeated benefits that it can provide to multiple inferences to static graphs. The processed graph is then fed to a hardware accelerator implemented in a FPGA consisting of differentiated pipelined modules for aggregation (sparse) and combination (dense systolic array + non-linear activation module).

As differentiating elements with respect to other designs, we find that the aggregator module uses a double-buffering technique to hide latency of additions, and exploits both node-level and feature-level parallelism. We also observe that the accelerator implements two modes of operation depending on the order of the matrix multiplications, which leads to different strategies for pipelining. To accommodate them, the modules are interconnected both from the aggregate module to the combination modules, and *vice versa*.

GraphACT. While all other accelerators focused on inference, GraphACT [119] explores how to efficiently perform GNN training in an heterogeneous CPU+FPGA platform. The main design decision relates to deciding which parts are computed where and which data to store in memory. To these questions, the authors argue that CPU performs graph sampling and the calculation of the loss gradients, while and the FPGA does forward and backward propagation passes. The FPGA, thus implements aggregation and combination. The authors present optimizations based on the scheduling of the different operations taking into consideration that backpropagation can be performed after batching of multiple layers or batching different parts of the graph. Moreover, similarly to in [116], redundant operations at aggregation are eliminated via searching of edges common to multiple vertices.

V. GNN Acceleration: The Vision

Previous sections have discussed how GNNs can be understood as a set of classical NNs working symbiotically over graph-structured data. We have seen that, to extract specific knowledge from the graphs, different NN layers may be employed leading to a wide variety of GNN flavours. This, plus the fundamental dependence of GNNs on the input graph (which may be extremely large) complicate the task of streamlining their execution enormously. As a result, works that have emerged trying to accelerate GNNs have implicitly made a choice upon either providing an extremely efficient acceleration scheme for a specific GNN variant, or being general or flexible enough to serve multiple types of GNNs less efficiently.

The key challenge in GNN acceleration is thus to deliver an operational and architectural framework that is able to both maximize performance and efficiency while maintaining a degree of flexibility that caters to the different graph sizes, characteristics, and GNN algorithms. Albeit a daunting task, in this section we aim to leverage the analysis of existing acceleration works to hypothesize which would be the main characteristics that future GNN accelerators should feature. In particular, our envisaged architectural approach shall be driven by (i) software-hardware co-design, (ii) graph awareness, and (iii) an much-needed emphasis on communications. We next discuss these aspects qualitatively, using Figure 10 as reference.

A. *Software-Hardware Co-Design*

The analysis of prior work has shown that both software and hardware approaches can provide significant speedups. In some occasions, one might argue that both strategies attack the problem similarly, e.g. node reordering in software [123] and workload balancing in hardware [42]. However, a few works have also started to realize that both approaches are not mutually exclusive and that their benefits can add up, or one can simplify the other. For instance, the design from Zhang *et al.* [116] eliminates redundant operations via software pre-processing and then optimizes execution with specialized modules for aggregation and combination. The software side allows to avoid having specialized hardware structures to eliminate redundant operations.

Building upon this observation, our first proposed pillar is *software-hardware co-design* as a strategy for handling different GNNs and graphs efficiently while retaining some hardware simplicity. We advocate for a control-data plane model where, in general, the control plane will be implemented entirely in software providing the flexibility and the data plane will be implemented in custom hardware providing the efficiency. While conceptually separated (see Fig. 10), the operation of both planes will be tightly coupled.

On the one hand, the **control plane** manages the actions of the accelerator by having a global view of the complete GNN structure and input graph. The control plane is responsible for dictating the dataflow running in the data plane, by (i) partitioning the GNN computation into manageable computational segments, (ii) mapping the different vertices and edges to the hardware resources of the data plane, and (iii) scheduling the different executions towards balancing the workload, maximize the benefits of pipelining, and so on. Finally, we also consider part of the control plane to (iv) drive pre-processing (and possibly offline) steps such as the removal of redundant operations [51] or the calculation of certain graph aspects such as community detection [39]. By being implemented in software, these functions can deliver the required flexibility to accelerate any GNN workload.

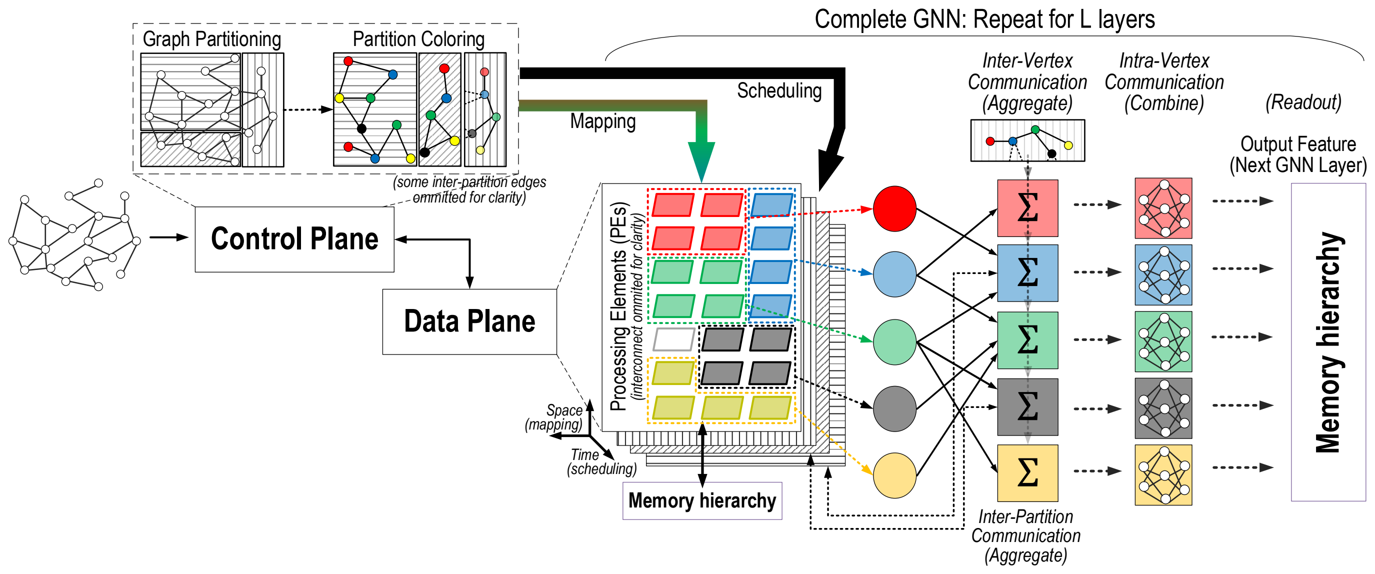
Complete GNN: Repeat for L layers
Graph Partitioning
Partition Coloring
(some inter-partition edges ommitted for clarity)
Scheduling
Mapping
Control Plane
Data Plane
Processing Elements (PEs)
(interconnect ommited for clarity)
Space (mapping)
Time (scheduling)
Memory hierarchy
Inter-Vertex Communication (Aggregate)
Intra-Vertex Communication (Combine)
(Readout)
Output Feature (Next GNN Layer)
Memory hierarchy
Inter-Partition Communication (Aggregate)

Fig. 10: Architectural vision for GNN accelerators with hardware-software co-design (i.e. control and data planes), graph awareness (i.e. guided mapping and scheduling), and communication-centric design (i.e. reconfigurable interconnect).

On the other hand, the **data plane** consists of the processing and memory elements that work as per the control plane instructions to execute a GNN. As we have seen in Section IV-B, we could adopt many strategies for architecting the data plane, e.g., unified, phased, modular, homogenous, heterogeneous, to name a few. However, we find particularly interesting the use of architectures similar to that of MAERI [33], where an homogeneous array of Processing Elements (PEs) and a specialized memory hierarchy are put together via a lightweight reconfigurable interconnect fabric. This fabric adapts the dataflow according to the control plane commands, thus allowing to give service to the multiple execution stages of an algorithm or different algorithms.

B. *Graph Awareness*

Most accelerators have attempted to provide methods that adapt to runtime conditions while being largely unaware of the input graph characteristics [40], [124]. However, it has been also realized that aspects such as the size of the graph, the relative size of the feature vectors, the clustering factor of the graph, or the average degree of the same can be extremely relevant in accelerating the GNN [41], [47], [123]. In fact, GNNAdvisor [39] seeks to exploit this information explicitly to improve the performance in GPUs, while others have based the order of operations or the mapping of PEs on characterstics of the graph [40]–[42].

This leads to the second pillar of our envisaged architecture: *graph awareness*. If the GNN depends on the input graph, then maximizing performance needs to be aware of the main features of that graph. Offline or online methods shall be used to extract useful information from the graph that, in our case, will be leveraged by the control plane. This will thus affect aspects such as the graph partitioning [123], which may be more or less aggressive depending on the degree distribution; the ordering and pipelining of the different aggregate–combine phases, which may vary across layers and across graphs; or the scheduling process to minimize inter-partition communication. A good example of this is found in community detection, whose efficient implementation [175], [176] or prediction [96] may allow for the partition of the graph in densely connected graphlets at runtime. This is relevant to efficient pooling [154], redundancy elimination [51], and optimal scheduling [39].

C. *Communication-Centric Design*

Data movement is the enemy of efficient architectures. Hardware accelerators aim to minimize it by adapting its resources to the execution dataflow but, surprisingly, traditional DNN accelerators [34], [160] have generally given a relatively low importance to the sub-system handling data movement: the interconnect fabric. This is also true for GNN accelerators, which are generally computing-centric with few exceptions [37], [119]. However, GNNs pose the additional challenge of not having a single optimal dataflow given the input graph dependence and the many algorithm variants. Thus, data movement continues to be a crucial aspect.

For this reason, the third pillar of our envisaged architecture is taking a *communication-centric design* approach. This is a philosophy that has been applied to endow DNN accelerators with certain flexibility [33], [177], [178] or to optimize distributed learning [179]. In our case, we propose the use of a reconfigurable interconnect fabric among the PEs to adapt the hardware to the underlying graph connectivity or, in other words, to the optimal dataflow that may vary across layers, partitions, or graphs. In an extreme case, one could adopt the approach of recent DNN accelerators that orchestrate all data movement at compilation time [180], [181]. GNNs and their extreme size might discourage the use of this strategy and, instead, advocate for a compilation that provides hints for the interconnect to adapt to the varying needs of the graph and its most optimal dataflow. The compilation and reconfiguration could be complemented by the analysis of the input graph, which would allow us to predict the prevalent communication patterns and, thus, the most appropriate interconnect topology.

VI. Conclusion

The recent interest in geometric deep learning, or methods able to model and predict graph-structured data, have led to an explosion of research around GNNs. As we have seen in our analysis of the current state of the art, most of the works focus on the algorithms and their applications, rendering the topic of GNN computing a less beaten path. However, we anticipate that the area of software and hardware support for GNNs will grow at a fast pace, continuing an upwards trend that we observed from 2018 to today.

The reasons for the probable increase in research delving into more efficient computing means for GNNs are several. First, the field is maturing and the more theoretical algorithm-driven research gives way to the most application-oriented development. A clear example of this trend is the apparition of efforts to unify aspects such as benchmarking [167]. Second, GNNs are the key to many disruptive applications in multiple fields, thus creating a clear application pull driving the need for better processing. Third, GNNs present multiple unique challenges such as the wide variety of algorithm variants, their dependence on the graph characteristics, or their massive scale in some applications. This makes the field of GNN processing unlikely to saturate in the foreseeable future.

Finally, we highlight the rising popularity of software frameworks and the recent apparition of hardware accelerators for GNNs. On the software side, libraries such as DGL or NeuGraph aim to speed up and add features to widespread frameworks such as TF or PyTorch. Interesting contributions are acceleration of GNNs via graph analysis or pre-coding, as well as the distribution of computation in large-scale systems, much needed for huge recommendation systems. On the hardware side, we did not observe a clear architectural trend and existing proposals are debating between being specific or applicable to multiple GNN variants, and between unified architectures or more hierarchical, tiled organizations. Building on this observation, we envision that future accelerators shall adopt a hardware-software co-design approach to maximize performance, keep graph awareness as a profitable optimization opportunity, and tackle workload variability via a reconfigurable interconnect.

References

[1] Y. Lecun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[2] K. He *et al.*, "Deep Residual Learning for Image Recognition," in *CVPR*, 2016.

[3] T. Young, D. Hazarika, S. Poria, and E. Cambria, "Recent trends in deep learning based natural language processing [review article]," *IEEE Computational Intelligence Magazine*, vol. 13, no. 3, pp. 55–75, 2018.

[4] A. Esteva, B. Kuprel, R. A. Novoa, J. Ko, S. M. Swetter, H. M. Blau, and S. Thrun, "Dermatologist-level classification of skin cancer with deep neural networks.," *Nature*, vol. 542, no. 7639, pp. 115–118, 2017.

[5] E. Guresen, G. Kayakutlu, and T. U. Daim, "Using artificial neural network models in stock market index prediction," *Expert Systems with Applications*, vol. 38, no. 8, pp. 10389 – 10397, 2011.

[6] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, C. Gulcehre, F. Song, A. Ballard, J. Gilmer, G. Dahl, A. Vaswani, K. Allen, C. Nash, V. Langston, C. Dyer, N. Heess, D. Wierstra, P. Kohli, M. Botvinick, O. Vinyals, Y. Li, and R. Pascanu, "Relational inductive biases, deep learning, and graph networks," *arXiv preprint arXiv:1806.01261*, 2018.

[7] M. Botvinick, S. Ritter, J. X. Wang, Z. Kurth-Nelson, C. Blundell, and D. Hassabis, "Reinforcement Learning, Fast and Slow," *Trends in Cognitive Sciences*, vol. 23, no. 5, pp. 408–422, 2019.

[8] J. Wang, Y. Yang, J. Mao, Z. Huang, C. Huang, and W. Xu, "CNN-RNN: A unified framework for multi-label image classification," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2285–2294, 2016.

[9] M. M. Bronstein, J. Bruna, Y. Lecun, A. Szlam, and P. Vandergheynst, "Geometric Deep Learning: Going beyond Euclidean data," *IEEE Signal Process. Mag.*, vol. 34, no. 4, pp. 18–42, 2017.

[10] Z. Zhang, P. Cui, and W. Zhu, "Deep Learning on Graphs: A Survey," *IEEE Trans. Knowl. Data Eng.*, vol. 14, no. 8, pp. 1–1, 2020.

[11] K. Rusek, J. SuÃąrez-Varela, A. Mestres, P. Barlet-Ros, and A. Cabellos-Aparicio, "Unveiling the potential of Graph Neural Networks for network modeling and optimization in SDN," *Proceedings of the 2019 ACM Symposium on SDN Research - SOSR âĂŹ19*, 2019.

[12] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," *34th Int. Conf. Mach. Learn. ICML 2017*, vol. 3, pp. 2053–2070, apr 2017.

[13] D. F. Nettleton, "Data mining of social networks represented as graphs," *Comput. Sci. Rev.*, vol. 7, pp. 1–34, 2013.

[14] X. Li, Y. Zhou, S. Gao, N. Dvornek, M. Zhang, J. Zhuang, S. Gu, D. Scheinost, L. Staib, P. Ventola, and J. Duncan, "Braingnn: Interpretable brain graph neural network for fmri analysis," *bioRxiv*, 2020.

[15] M. Gori, G. Monfardini, and F. Scarselli, "A new model for learning in Graph domains," *Proceedings of the International Joint Conference on Neural Networks*, vol. 2, pp. 729–734, 2005.

[16] F. Scarselli, M. Gori, Ah Chung Tsoi, M. Hagenbuchner, and G. Monfardini, "The Graph Neural Network Model," *IEEE Trans. Neural Networks*, vol. 20, pp. 61–80, jan 2009.

[17] A. Fout, J. Byrd, B. Shariat, and A. Ben-Hur, "Protein interface prediction using graph convolutional networks," *Advances in Neural Information Processing Systems*, vol. 2017-Decem, no. Nips, pp. 6531–6540, 2017.

[18] W. Fan, Y. Ma, Q. Li, Y. He, E. Zhao, J. Tang, and D. Yin, "Graph neural networks for social recommendation," *The Web Conference 2019 - Proceedings of the World Wide Web Conference, WWW 2019*, pp. 417–426, 2019.

[19] S. Vashishth, *Neural Graph Embedding Methods for Natural Language Processing*. PhD thesis, IISc Bangalore, 2020.

[20] Z. Xie, W. Lv, S. Huang, Z. Lu, and B. Du, "Sequential Graph Neural Network for Urban Road Traffic Speed Prediction," *IEEE Access*, vol. 8, pp. 63349–63358, 2020.

[21] S. L. Yong, M. Hagenbuchner, A. C. Tsoi, F. Scarselli, and M. Gori, "Document mining using graph neural network," in *Comparative Evaluation of XML Information Retrieval Systems* (N. Fuhr, M. Lalmas, and A. Trotman, eds.), (Berlin, Heidelberg), pp. 458–472, Springer Berlin Heidelberg, 2007.

[22] X. Wang, R. Girshick, A. Gupta, and K. He, "Non-local Neural Networks," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 7794–7803, 2018.

[23] K. Rusek and P. Cholda, "Message-Passing Neural Networks Learn Little's Law," *IEEE Communications Letters*, vol. 23, pp. 274–277, feb 2019.

[24] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph neural networks: A review of methods and applications," *arXiv preprint arXiv:1812.08434*, 2018.

[25] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A Comprehensive Survey on Graph Neural Networks," *IEEE Trans. Neural Networks Learn. Syst.*, pp. 1–21, 2020.

[26] L. C. Lamb, A. Garcez, M. Gori, M. Prates, P. Avelar, and M. Vardi, "Graph neural networks meet neural-symbolic computing: A survey and perspective," *arXiv preprint arXiv:2003.00330*, 2020.

[27] W. L. Hamilton, R. Ying, and J. Leskovec, "Representation learning on graphs: Methods and applications," *arXiv preprint arXiv:1709.05584*, 2017.

[28] P. Cui, X. Wang, J. Pei, and W. Zhu, "A Survey on Network Embedding," *IEEE Trans. Knowl. Data Eng.*, vol. 31, no. 5, pp. 833–852, 2019.

[29] J. B. Lee, R. A. Rossi, S. Kim, N. K. Ahmed, and E. Koh, "Attention models in graphs: A survey," *ACM Trans. Knowl. Discov. Data*, vol. 13, no. 6, 2019.

[30] I. Brugere, B. Gallagher, and T. Y. Berger-Wolf, "Network structure inference, a survey: Motivations, methods, and applications," *ACM Comput. Surv.*, vol. 51, no. 2, 2018.

[31] Y. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, 2017.

[32] S. Mittal, "A survey of FPGA-based accelerators for convolutional neural networks," *Neural Comput. & Appl.*, vol. 32, pp. 1109–1139, 2020.

[33] H. Kwon, A. Samajdar, and T. Krishna, "Maeri: Enabling flexible dataflow mapping over dnn accelerators via reconfigurable interconnects," in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS âĂŽ18, (New York, NY, USA), p. 461âĂŞ475, Association for Computing Machinery, 2018.

[34] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam, "Shidiannao: Shifting vision processing closer to the sensor," in *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*, pp. 92–104, 2015.

[35] K. Hegde, J. Yu, R. Agrawal, M. Yan, M. Pellauer, and C. Fletcher, "Ucnn: Exploiting computational reuse in deep neural networks via weight repetition," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pp. 674–687, 2018.

[36] Y. Chen, T. Yang, J. Emer, and V. Sze, "Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 2, pp. 292–308, 2019.

[37] A. Tripathy, K. Yelick, and A. Buluc, "Reducing communication in graph neural network training," *arXiv preprint arXiv:2005.03300*, 2020.

[38] L. Ma, Z. Yang, Y. Miao, J. Xue, M. Wu, L. Zhou, and Y. Dai, "Neugraph: Parallel deep neural network computation on large graphs," in *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, pp. 443–458, 2019.

[39] Y. Wang, B. Feng, G. Li, S. Li, L. Deng, Y. Xie, and Y. Ding, "Gnnadvisor: An efficient runtime system for gnn acceleration on gpus," *arXiv preprint arXiv:2006.06608*, 2020.

[40] S. Liang, Y. Wang, C. Liu, L. He, H. Li, and X. Li, "EnGN: A High-Throughput and Energy-Efficient Accelerator for Large Graph Neural Networks," *IEEE Transactions on Computers*, 2020.

[41] M. Yan *et al.*, "Hygcn: A gcn accelerator with hybrid architecture," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 15–29, 2020.

[42] T. Geng, A. Li, T. Wang, C. Wu, Y. Li, A. Tumeo, S. Che, S. Reinhardt, and M. Herbordt, "Awb-gcn: A graph convolutional network accelerator with runtime workload rebalancing," *arXiv preprint arXiv:1908.10834*, 2019.

[43] A. Auten, M. Tomei, and R. Kumar, "Hardware Acceleration of Graph Neural Networks," *DAC*, 2020.

[44] S. Dave, R. Baghdadi, T. Nowatzki, S. Avancha, A. Shrivastava, and B. Li, "Hardware acceleration of sparse and irregular tensor computations of ml models: A survey and insights," *arXiv preprint arXiv:2007.00864*, 2020.

[45] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *Advances in Neural Information Processing Systems*, vol. 2017-Decem, no. Nips, pp. 1025–1035, 2017.

[46] R. Liao, M. Brockschmidt, D. Tarlow, A. L. Gaunt, R. Urtasun, and R. S. Zemel, "Graph partition neural networks for semi-supervised classification," *6th International Conference on Learning Representations, ICLR 2018 - Workshop Track Proceedings*, mar 2018.

[47] M. Fey and J. E. Lenssen, "Fast graph representation learning with pytorch geometric," *arXiv preprint arXiv:1903.02428*, 2019.

[48] M. Wang, L. Yu, D. Zheng, Q. Gan, Y. Gai, Z. Ye, M. Li, J. Zhou, Q. Huang, C. Ma, *et al.*, "Deep graph library: Towards efficient and scalable deep learning on graphs," *arXiv preprint arXiv:1909.01315*, 2019.

[49] K. Kiningham, P. Levis, and C. Re, "GReTA: Hardware Optimized Graph Processing for GNNs," in *Proceedings of the Workshop on Resource-Constrained Machine Learning (ReCoML 2020)*, March 2020.

[50] M. Yan, Z. Chen, L. Deng, X. Ye, Z. Zhang, D. Fan, and Y. Xie, "Characterizing and understanding gcns on gpu," *IEEE Computer Architecture Letters*, vol. 19, no. 1, pp. 22–25, 2020.

[51] Z. Jia, S. Lin, R. Ying, and A. Aiken, "Redundancy-Free Computation for Graph Neural Networks," in *Proceedings of the KDD '20*, 2020.

[52] Q. Li, Z. Han, and X. M. Wu, "Deeper insights into graph convolutional networks for semi-supervised learning," *32nd AAAI Conf. Artif. Intell. AAAI 2018*, pp. 3538–3545, 2018.

[53] K. Xu, S. Jegelka, W. Hu, and J. Leskovec, "How powerful are graph neural networks?," *7th Int. Conf. Learn. Represent. ICLR 2019*, pp. 1–17, 2019.

[54] F. J. Pineda, "Generalization of Back-Propagation to Recurrent Neural Networks," *Phys. Rev. Lett.*, vol. 59, no. 19, pp. 2229–2232, 1987.

[55] L. B. Almeida, "A learning rule for asynchronous perceptrons with feedback in a combinatorial environment," in *Artificial neural networks: concept learning*, pp. 102–111, 1990.

[56] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph convolutional neural networks for web-scale recommender systems," *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Min.*, pp. 974–983, 2018.

[57] D. Tkaczyk, P. Szostek, M. Fedoryszak, P. J. Dendek, and Ł. Bolikowski, "Cermine: automatic extraction of structured metadata from scientific literature," *International Journal on Document Analysis and Recognition (IJDAR)*, vol. 18, no. 4, pp. 317–335, 2015.

[58] J. Webber, "A programmatic introduction to neo4j," in *Proceedings of the 3rd annual conference on Systems, programming, and applications: software for humanity*, pp. 217–218, 2012.

[59] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *5th Int. Conf. Learn. Represent. ICLR 2017 - Conf. Track Proc.*, sep 2019.

[60] X. Jiang, P. Ji, and S. Li, "CensNet: Convolution with edge-node switching in graph neural networks," *IJCAI International Joint Conference on Artificial Intelligence*, vol. 2019-Augus, pp. 2656–2662, 2019.

[61] D. Oñoro-Rubio, M. Niepert, A. García-Durán, R. González, and R. J. López-Sastre, "Representation Learning for Visual-Relational Knowledge Graphs," *ArXiv e-prints*, 2017.

[62] M. Niepert, M. Ahmad, and K. Kutzkov, "Learning convolutional neural networks for graphs," *33rd Int. Conf. Mach. Learn. ICML 2016*, vol. 4, pp. 2958–2967, 2016.

[63] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. van den Berg, I. Titov, and M. Welling, "Modeling Relational Data with Graph Convolutional Networks," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 10843 LNCS, pp. 593–607, mar 2018.

[64] A. G. Duran and M. Niepert, "Learning graph representations with embedding propagation," in *Advances in neural information processing systems*, pp. 5119–5130, 2017.

[65] T. Hamaguchi, H. Oiwa, M. Shimbo, and Y. Matsumoto, "Knowledge transfer for out-of-knowledge-base entities: A graph neural network approach," *arXiv preprint arXiv:1706.05674*, 2017.

[66] M. Henaff, J. Bruna, and Y. LeCun, "Deep convolutional networks on graph-structured data," *arXiv preprint arXiv:1506.05163*, 2015.

[67] V. P. Dwivedi, C. K. Joshi, T. Laurent, Y. Bengio, and X. Bresson, "Benchmarking graph neural networks," *arXiv preprint arXiv:2003.00982*, 2020.

[68] F. Scarselli, M. Hagenbuchner, S. L. Yong, A. C. Tsoi, M. Gori, and M. Maggini, "Graph neural networks for ranking web pages," *Proceedings - 2005 IEEE/WIC/ACM InternationalConference on Web Intelligence, WI 2005*, vol. 2005, pp. 666–672, 2005.

[69] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph Convolutional Neural Networks for Web-Scale Recommender Systems," *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Jul 2018.

[70] D. Duvenaud, D. Maclaurin, J. Aguilera-Iparraguirre, R. Gómez-Bombarelli, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams, "Convolutional networks on graphs for learning molecular fingerprints," *Adv. Neural Inf. Process. Syst.*, pp. 2224–2232, 2015.

[71] V. Myska, R. Burget, and B. Peter, "Graph neural network for website element detection," *2019 42nd International Conference on Telecommunications and Signal Processing, TSP 2019*, pp. 216–219, 2019.

[72] P. C. St John, C. Phillips, T. W. Kemper, A. N. Wilson, Y. Guan, M. F. Crowley, M. R. Nimlos, and R. E. Larsen, "Message-passing neural networks for high-throughput polymer screening," *Journal of Chemical Physics*, vol. 150, jun 2019.

[73] B. Sanchez-Lengeling *et al.*, "Machine learning for scent: Learning generalizable perceptual representations of small molecules," *arXiv preprint arXiv:1910.10685*, 2019.

[74] G. Monfardini, V. Di Massa, F. Scarselli, and M. Gori, "Graph neural networks for object localization," *Frontiers in Artificial Intelligence and Applications*, vol. 141, no. August 2019, pp. 665–669, 2006.

[75] M. Zhang and Y. Chen, "Link prediction based on graph neural networks," *Advances in Neural Information Processing Systems*, vol. 2018-Decem, no. NeurIPS, pp. 5165–5175, 2018.

[76] M. Verma and D. Ganguly, "Graph Edit Distance Computation via Graph Neural Networks Yunsheng," *SIGIR 2019 - Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 1281–1284, 2019.

[77] R. Li, M. Tapaswi, R. Liao, J. Jia, R. Urtasun, and S. Fidler, "Situation Recognition with Graph Neural Networks," *Proceedings of the IEEE International Conference on Computer Vision*, vol. 2017-Octob, pp. 4183–4192, 2017.

[78] Y. Ma, H. Ren, B. Khailany, H. Sikka, L. Luo, K. Natarajan, and B. Yu, "High performance graph convolutional networks with applications in testability analysis," *Proceedings - Design Automation Conference*, 2019.

[79] D. Zhu and Y. Liu, "Modelling spatial patterns using graph convolutional networks," *Leibniz Int. Proc. Informatics, LIPIcs*, vol. 114, pp. 1–11, 2018.

[80] A. Sanchez-Gonzalez, N. Heess, J. T. Springenberg, J. Merel, M. Riedmiller, R. Hadsell, and P. Battaglia, "Graph networks as learnable physics engines for inference and control," *35th International Conference on Machine Learning, ICML 2018*, vol. 10, pp. 7097–7117, 2018.

[81] P. Battaglia *et al.*, "Interaction networks for learning about objects, relations and physics," *Advances in neural information processing systems*, pp. 4502–4510, 2016.

[82] Z. Cui, K. Henrickson, R. Ke, and Y. Wang, "Traffic Graph Convolutional Recurrent Neural Network: A Deep Learning Framework for Network-Scale Traffic Learning and Forecasting," *IEEE Transactions on Intelligent Transportation Systems*, pp. 1–12, 2019.

[83] V. Zayats and M. Ostendorf, "Conversation modeling on reddit using a graph-structured lstm," *Transactions of the Association for Computational Linguistics*, vol. 6, pp. 121–132, 2018.

[84] S. Guo, Y. Lin, N. Feng, C. Song, and H. Wan, "Attention based spatial-temporal graph convolutional networks for traffic flow forecasting," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, pp. 922–929, 2019.

[85] J. Qiu, J. Tang, H. Ma, Y. Dong, K. Wang, and J. Tang, "DeepInf: Social Influence Prediction with Deep Learning," *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, Jul 2018.

[86] B.-H. Kim and J. C. Ye, "Understanding graph isomorphism network for brain mr functional connectivity analysis," *arXiv preprint arXiv:2001.03690*, 2020.

[87] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "Computational capabilities of graph neural networks," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 81–102, 2009.

[88] J. Chen, J. Zhu, and L. Song, "Stochastic training of graph convolutional networks with variance reduction," *35th International Conference on Machine Learning, ICML 2018*, vol. 3, pp. 1503–1532, oct 2018.

[89] M. Bianchini, M. Maggini, L. Sarti, and F. Scarselli, "Recursive neural networks for processing graphs with labelled edges: Theory and applications," *Neural Networks*, vol. 18, no. 8, pp. 1040–1050, 2005.

[90] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and deep locally connected networks on graphs," *2nd International Conference on Learning Representations, ICLR 2014 - Conference Track Proceedings*, pp. 1–14, 2014.

[91] J. Wu, J. He, and J. Xu, "Demo-Net: Degree-specific graph neural networks for node and graph classification," in *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 406–415, Association for Computing Machinery, jul 2019.

[92] Z. Ying, D. Bourgeois, J. You, M. Zitnik, and J. Leskovec, "Gnnexplainer: Generating explanations for graph neural networks," in *Advances in neural information processing systems*, pp. 9244–9255, 2019.

[93] T. Kawamoto, M. Tsubaki, and T. Obuchi, "Mean-field theory of graph neural networks in graph partitioning," *Advances in Neural Information Processing Systems*, vol. 2018-Decem, pp. 4361–4371, 2018.

[94] V. Di Massa, G. Monfardini, L. Sarti, F. Scarselli, M. Maggini, and M. Gori, "A comparison between recursive neural networks and graph neural networks," *IEEE International Conference on Neural Networks - Conference Proceedings*, pp. 778–785, 2006.

[95] D. Zügner and S. Günnemann, "Adversarial attacks on graph neural networks via meta learning," *7th International Conference on Learning Representations, ICLR 2019*, pp. 1–15, 2019.

[96] Z. Chen, J. Bruna, and L. Li, "Supervised community detection with line graph neural networks," *7th International Conference on Learning Representations, ICLR 2019*, pp. 1–23, 2019.

[97] S. Verma and Z. L. Zhang, "Stability and generalization of graph convolutional neural networks," *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, no. May, pp. 1539–1548, 2019.

[98] M. Nickel, K. Murphy, V. Tresp, and E. Gabrilovich, "A review of relational machine learning for knowledge graphs," *Proceedings of the IEEE*, vol. 104, no. 1, pp. 11–33, 2016.

[99] F. J. Pineda, "Generalization of Backpropagation To Recurrent and High-Order Networks.," p. 15, 1987.

[100] Y. Yang, X. Wang, M. Song, J. Yuan, and D. Tao, "Spagan: Shortest path graph attention network.," in *IJCAI*, pp. 4099–4105, 2019.

[101] J. Chen, T. Ma, and C. Xiao, "FastGCN: Fast learning with graph convolutional networks via importance sampling," *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*, jan 2018.

[102] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *arXiv preprint arXiv:1901.00596*, 2019.

[103] Y. Li, R. Zemel, M. Brockschmidt, and D. Tarlow, "Gated graph sequence neural networks," *4th Int. Conf. Learn. Represent. ICLR 2016 - Conf. Track Proc.*, nov 2016.

[104] T. N. Kipf and M. Welling, "Variational graph auto-encoders," *arXiv preprint arXiv:1611.07308*, 2016.

[105] N. Bandinelli, M. Bianchini, and F. Scarselli, "Learning long-term dependencies using layered graph neural networks," *Proceedings of the International Joint Conference on Neural Networks*, 2010.

[106] L. Zhang, D. Xu, A. Arnab, and P. H. Torr, "Dynamic graph message passing networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3726–3735, 2020.

[107] F. Monti, M. M. Bronstein, and X. Bresson, "Geometric matrix completion with recurrent multi-graph neural networks," *Advances in Neural Information Processing Systems*, vol. 2017-Decem, no. Nips, pp. 3698–3708, 2017.

[108] H. T. Son and C. Jones, "Graph neural networks with efficient tensor operations in cuda/gpu and graphflow deep learning framework in c++ for quantum chemistry," 2019.

[109] T. Ma and A. Zhang, "Affinitynet: semi-supervised few-shot learning for disease type prediction," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, pp. 1069–1076, 2019.

[110] H. Park and J. Neville, "Exploiting interaction links for node classification with deep graph neural networks.," in *IJCAI*, pp. 3223–3230, 2019.

[111] N. De Cao and T. Kipf, "Molgan: An implicit generative model for small molecular graphs," *arXiv preprint arXiv:1805.11973*, 2018.

[112] Y. Li, O. Vinyals, C. Dyer, R. Pascanu, and P. Battaglia, "Learning deep generative models of graphs," *arXiv preprint arXiv:1803.03324*, 2018.

[113] K. K. Thekumparampil *et al.*, "Attention-based graph neural network for semi-supervised learning," *arXiv preprint arXiv:1803.03735*, 2018.

[114] V. Garcia and J. Bruna, "Few-shot learning with graph neural networks," *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*, pp. 1–13, 2018.

[115] P. Veličković, A. Casanova, P. Liò, G. Cucurull, A. Romero, and Y. Bengio, "Graph attention networks," *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*, pp. 1–12, 2018.

[116] B. Zhang, H. Zeng, and V. Prasanna, "Hardware Acceleration of Large Scale GCN Inference," in *Proceedings of the ASAP '20*, pp. 61–68, 2020.

[117] K. Kiningham, C. Re, and P. Levis, "GRIP: A Graph Neural Network Accelerator Architecture," pp. 1–14, 2020.

[118] R. Zhu, K. Zhao, H. Yang, W. Lin, C. Zhou, B. Ai, Y. Li, and J. Zhou, "AliGraph: A comprehensive graph neural network platform," *Proceedings of the VLDB Endowment*, vol. 12, no. 12, pp. 2094–2105, 2018.

[119] H. Zeng and V. Prasanna, "GraphACT: Accelerating GCN training on CPU-FPGA heterogeneous platforms," *FPGA 2020 - 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 255–265, 2020.

[120] "Geom: A Hierarchical Spatial Architecture with Hybrid Dataflows for Graph Learning," pp. 1–13, 2020.

[121] L. Ma *et al.*, "Towards efficient large-scale graph neural network computing," *arXiv preprint arXiv:1810.08403*, 2018.

[122] M. Balog *et al.*, "Fast training of sparse graph neural networks on dense hardware," *arXiv preprint arXiv:1906.11786*, 2019.

[123] C. Tian, L. Ma, Z. Yang, and Y. Dai, "PCGCN: Partition-Centric Processing for Accelerating Graph Convolutional Network," *Proceedings of IPDPS 2020*, pp. 936–945, 2020.

[124] Z. Jia, S. Lin, M. Gao, M. Zaharia, and A. Aiken, "Improving the accuracy, scalability, and performance of Graph Neural Networks with ROC," in *Proceedings of MLSys '20*, 2020.

[125] R. C. Glen, A. Bender, C. H. Arnby, L. Carlsson, S. Boyer, and J. Smith, "Circular fingerprints: flexible molecular descriptors with applications from physical chemistry to adme," *IDrugs*, vol. 9, no. 3, p. 199, 2006.

[126] T. Horváth, T. Gärtner, and S. Wrobel, "Cyclic pattern kernels for predictive graph mining," in *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 158–167, 2004.

[127] P. Mahé and J.-P. Vert, "Graph kernels based on tree patterns for molecules," *Machine learning*, vol. 75, no. 1, pp. 3–35, 2009.

[128] T. Gärtner, P. Flach, and S. Wrobel, "On graph kernels: Hardness results and efficient alternatives," in *Learning Theory and Kernel Machines* (B. Schölkopf and M. K. Warmuth, eds.), (Berlin, Heidelberg), pp. 129–143, Springer Berlin Heidelberg, 2003.

[129] S. Ghosh, N. Das, T. Gonçalves, and P. Quaresma, "The journey of graph kernels through two decades," *Comput. Sci. Rev.*, vol. 27, pp. 88–111, 2018.

[130] B. Schlkopf, A. J. Smola, and F. Bach, *Learning with kernels: support vector machines, regularization, optimization, and beyond.* the MIT Press, 2018.

[131] J. Atwood and D. Towsley, "Diffusion-convolutional neural networks," in *Advances in neural information processing systems*, pp. 1993–2001, 2016.

[132] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Advances in neural information processing systems*, pp. 3844–3852, 2016.

[133] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk," *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD âĂŹ14*, 2014.

[134] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," 2016.

[135] A. Narayanan, M. Chandramohan, R. Venkatesan, L. Chen, Y. Liu, and S. Jaiswal, "graph2vec: Learning distributed representations of graphs," 2017.

[136] S. Sukhbaatar, R. Fergus, *et al.*, "Learning multiagent communication with backpropagation," in *Advances in neural information processing systems*, pp. 2244–2252, 2016.

[137] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," *arXiv preprint arXiv:1409.1259*, 2014.

[138] W.-L. Chiang, X. Liu, S. Si, Y. Li, S. Bengio, and C.-J. Hsieh, "Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 257–266, 2019.

[139] F. Wu, T. Zhang, A. H. de Souza Jr., C. Fifty, T. Yu, and K. Q. Weinberger, "Simplifying graph convolutional networks," 2019.

[140] F. Hu, Y. Zhu, S. Wu, L. Wang, and T. Tan, "Hierarchical graph convolutional networks for semi-supervised node classification," 2019.

[141] B. Yu, H. Yin, and Z. Zhu, "Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting," *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, Jul 2018.

[142] S. Yan, Y. Xiong, and D. Lin in *Spatial Temporal Graph Convolutional Networks for Skeleton-Based Action Recognition*, 2018.

[143] R. Li, S. Wang, F. Zhu, and J. Huang, "Adaptive graph convolutional neural networks," in *AAAI Conference on Artificial Intelligence*, 2018.

[144] C. Zhuang and Q. Ma, "Dual graph convolutional networks for graph-based semi-supervised classification," pp. 499–508, 04 2018.

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

[146] D. V. Tran, N. Navarin, and A. Sperduti, "On filter size in graph convolutional networks," 2018.

[147] X. Bresson and T. Laurent, "Residual gated graph convnets," *arXiv preprint arXiv:1711.07553*, 2017.

[148] G. Salha, R. Hennequin, and M. Vazirgiannis, "Keep it simple: Graph autoencoders without graph convolutional networks," *arXiv preprint arXiv:1910.00942*, 2019.

[149] D. Zhou, L. Zheng, J. Xu, and J. He, "Misc-gan: A multi-scale generative model for graphs," *Frontiers in Big Data*, vol. 2, p. 3, 04 2019.

[150] J. Leskovec, D. Chakrabarti, J. Kleinberg, C. Faloutsos, and Z. Ghahramani, "Kronecker graphs: An approach to modeling networks," *J. Mach. Learn. Res.*, vol. 11, p. 985âĂŞ1042, Mar. 2010.

[151] W. Liu, H. Cooper, M. H. Oh, S. Yeung, P.-Y. Chen, T. Suzumura, and L. Chen, "Learning graph topological features via gan," 2017.

[152] J. You, R. Ying, X. Ren, W. L. Hamilton, and J. Leskovec, "Graphrnn: Generating realistic graphs with deep auto-regressive models," *arXiv preprint arXiv:1802.08773*, 2018.

[153] Z. Chen *et al.*, "Bridging the gap between spatial and spectral domains: A survey on graph neural networks," *arXiv preprint arXiv:2002.11867*, 2020.

[154] Z. Ying, J. You, C. Morris, X. Ren, W. Hamilton, and J. Leskovec, "Hierarchical graph representation learning with differentiable pooling," in *Advances in neural information processing systems*, pp. 4800–4810, 2018.

[155] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention Is All You Need," *Nips*, 2017.

[156] B. Y. Weisfeiler and A. A. Leman, "A reduction of a graph to a canonical form and an algebra arising during this reduction.," *Nauchno-Technicheskaya Informatsia , 2(9)*, pp. 2–16, 1968.

[157] Z. Zhang, C. Niu, P. Cui, B. Zhang, W. Cui, and W. Zhu, "A simple and general graph neural network with stochastic message passing," 2020.

[158] A. Rahimi, T. Cohn, and T. Baldwin, "Semi-supervised user geolocation via graph convolutional networks," *ArXiv*, vol. abs/1804.08049, 2018.

[159] Z. Zhang, J. Leng, L. Ma, Y. Miao, C. Li, and M. Guo, "Architectural Implications of Graph Neural Networks," *IEEE Computer Architecture Letters*, vol. 19, no. 1, pp. 59–62, 2020.

[160] V. Sze, Y. Chen, T. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[161] C.-Y. Gui, L. Zheng, B. He, C. Liu, X.-Y. Chen, X.-F. Liao, and H. Jin, "A survey on graph processing accelerators: Challenges and opportunities," *Journal of Computer Science and Technology*, vol. 34, no. 2, pp. 339–371, 2019.

[162] "Build Graph Nets in Tensorflow," 2019.

[163] D. Grattarola and C. Alippi, "Graph neural networks in tensorflow and keras with spektral," *arXiv preprint arXiv:2006.12138*, 2020.

[164] Z. Wang, Y. Guan, G. Sun, D. Niu, Y. Wang, H. Zheng, and Y. Han, "Gnn-pim: A processing-in-memory architecture for graph neural networks," in *Conference on Advanced Computer Architecture*, pp. 73–86, Springer, 2020.

[165] Y. Wang, A. Davidson, Y. Pan, Y. Wu, A. Riffel, and J. D. Owens, "Gunrock: A high-performance graph processing library on the gpu," in *Proceedings of the 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pp. 1–12, 2016.

[166] G. Huang, G. Dai, Y. Wang, and H. Yang, "Ge-spmm: General-purpose sparse matrix-matrix multiplication on gpus for graph neural networks," *arXiv preprint arXiv:2007.03179*, 2020.

[167] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec, "Open graph benchmark: Datasets for machine learning on graphs," *arXiv preprint arXiv:2005.00687*, 2020.

[168] D. Zhang, X. Huang, Z. Liu, Z. Hu, X. Song, Z. Ge, Z. Zhang, L. Wang, J. Zhou, and Y. Qi, "Agl: a scalable system for industrial-purpose graph machine learning," *arXiv preprint arXiv:2003.02454*, 2020.

[169] Z. Jia, M. Zaharia, and A. Aiken, "Beyond data and model parallelism for deep neural networks," *SysML 2019*, 2019.

[170] Y. Hu, Z. Ye, M. Wang, J. Yu, D. Zheng, M. Li, Z. Zhang, Z. Zhang, and Y. Wang, "Featgraph: A flexible and efficient backend for graph neural network systems," *arXiv preprint arXiv:2008.11359*, 2020.

[171] A. Lerer, L. Wu, J. Shen, T. Lacroix, L. Wehrstedt, A. Bose, and A. Peysakhovich, "PyTorch-BigGraph: A Large-scale Graph Embedding System," in *Proceedings of MLSys '19*, 2019.

[172] Y. Nagasaka, A. Nukada, R. Kojima, and S. Matsuoka, "Batched sparse matrix multiplication for accelerating graph convolutional networks," *Proceedings - 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid 2019*, pp. 231–240, 2019.

[173] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[174] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM Journal on scientific Computing*, vol. 20, no. 1, pp. 359–392, 1998.

[175] S. Fortunato, "Community detection in graphs," *Physics Reports*, vol. 486, no. 3, pp. 75 – 174, 2010.

[176] F. D. Malliaros and M. Vazirgiannis, "Clustering and community detection in directed networks: A survey," *Physics Reports*, vol. 533, no. 4, pp. 95 – 142, 2013. Clustering and Community Detection in Directed Networks: A Survey.

[177] H. Kwon, A. Samajdar, and T. Krishna, "A Communication-centric Approach for Designing Flexible DNN Accelerators," *IEEE Micro*, vol. 38, no. 6, pp. 25–35, 2018.

[178] H. Kwon and T. Krishna, "Rethinking NoCs for Spatial Neural Network Accelerators," in *Proceedings of the NoCS '17*, p. Art. 19, 2017.

[179] Z. Tang, S. Shi, X. Chu, W. Wang, and B. Li, "Communication-Efficient Distributed Deep Learning : A Comprehensive Survey," *arXiv preprint arXiv:2003.06307v1*, no. 1, pp. 1–23, 2020.

[180] M. James, M. Tom, P. Groeneveld, and V. Kibardin, "Physical mapping of neural networks on a wafer-scale deep learning accelerator," in *Proceedings of the 2020 International Symposium on Physical Design*, pp. 145–149, 2020.

[181] D. Abts, J. Ross, J. Sparling, M. Wong-VanHaren, M. Baker, T. Hawkins, A. Bell, J. Thompson, T. Kahsai, G. Kimmell, *et al.*, "Think fast: a tensor streaming processor (TSP) for accelerating deep learning workloads," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pp. 145–158, IEEE, 2020.