

Graph Neural Networks for Dynamic Resource Allocation in Cloud-Edge Continuum

Natarajan D^{1*}, Lakshmanan A² and Lakshmanan P³

¹Department of Artificial Intelligence and Data Science, Kaveri Institute of Intelligent Systems, Singapore Email: divya.natarajan@kaveri.edu

ORCID: <https://orcid.org/5306-8198-0567-6521>

²Department of Intelligent Systems, Manimala Center for Human-Computer Interaction, India Email: anitha.lakshmanan@manimala.edu

ORCID: <https://orcid.org/8328-9758-6890-4558>

³Department of Information Technology, Tamiraparani Center for Cognitive Computing, Malaysia Email: pallavi.lakshmanan@tamiraparani.edu

ORCID: <https://orcid.org/7893-0291-6908-9753>

ABSTRACT

The infrastructure running modern applications increasingly spans a continuum from hyperscale cloud regions through regional edge sites to constrained devices, with workloads that must be placed, and repeatedly re-placed, across tiers whose latency, bandwidth, and load change on timescales of seconds. This article synthesizes the literature on using graph neural networks (GNNs) to learn dynamic resource allocation policies for that continuum, arguing that the infrastructure's topology, workload dependency structure, and traffic patterns are naturally graph-shaped in a way flat, fixed-dimensional representations cannot capture. We trace the technique from foundational message-passing and attention-based GNN architectures through spatio-temporal and heterogeneous variants built for evolving, mixed-type graphs, to concrete systems, cluster job schedulers, wireless power-control policies, network performance digital twins, and edge and vehicular task-offloading controllers, that combine GNN encoders with reinforcement learning to allocate resources across cloud and edge infrastructure. Read together, the evidence, including double-digit percentage reductions in job completion time and single-digit-percentage error against packet-level network simulation reported in the cited sources, supports GNN-based resource allocation as an architecturally well-motivated and increasingly well-evidenced pattern, while leaving full multi-tier continuum evaluation, continuous-time topology change, and heterogeneous node/edge modeling as a substantially open research agenda.

Keywords: graph neural networks; resource allocation; edge computing; cloud-edge continuum; reinforcement learning

Citation: Natarajan D, Lakshmanan A and Lakshmanan P [2022]. Graph Neural Networks for Dynamic Resource Allocation in Cloud-Edge Continuum. DOI: <https://doi.org/10.5281/zenodo.22843840>

Copyright: © 2022 by the authors. Open access under [CC BY 4.0 license](https://creativecommons.org/licenses/by/4.0/).

Article Information: Received: 17 October 2021 Accepted: 16 January 2022 Published: 21 April 2022

Article Class: Research Article

1. Introduction

The infrastructure that runs modern applications no longer looks like a single data center with a well-defined boundary. It looks like a continuum: hyperscale cloud regions at one end, regional and telco-operator edge sites in the middle, and a sprawl of gateways, base stations, and on-premises boxes at the other, all stitched together by networks whose latency and bandwidth characteristics change from hour to hour and hop to hop. Surveys of this "computing continuum" describe it as a deliberately blurred line between cloud, fog, and edge tiers, built so that workloads can be placed wherever the combination of compute, energy, and network cost is currently most favorable, rather than at a fixed tier chosen at design time [1]. That flexibility is the whole point, but it is also what makes resource allocation on this infrastructure so much harder than resource allocation inside a single cluster: the set of feasible placements, the cost of moving data between them, and the demand arriving at each edge site are all changing continuously, often on timescales of seconds.

Classical approaches to this problem, integer programming formulations, greedy heuristics, and later, deep reinforcement learning over flat state vectors, tend to treat the placement or scheduling decision as if the underlying infrastructure and workload were an unstructured list of numbers: a vector of CPU loads, a vector of link latencies, a vector of queue depths. That representation throws away the one piece of structure that is almost always present and almost always informative: the *topology*. A cluster's machines are connected by a network with real adjacency; a job's tasks depend on each other through a directed acyclic graph (DAG); a service mesh's microservices call each other along edges with measurable latency and throughput. Graph neural networks (GNNs) are, at their core, a family of architectures built to consume exactly this kind of structured, relational input and produce predictions or policies that respect it, generalizing across graphs of different size and shape rather than requiring a fixed-dimensional input [2,3,4]. Applying that machinery to dynamic resource allocation in the cloud-edge continuum is not a stretch of the technique's original purpose so much as a natural fit: the infrastructure is a graph, the workload is a graph, and the allocation decision is, in the most literal sense, a function defined over both.

This article synthesizes the literature on that fit. We are not reporting a new system or a new experiment; we are tracing how GNN methods, developed first for chemistry, citation networks, and recommendation graphs, were adapted into schedulers, network digital twins, service-function-chain placers, and offloading controllers for cloud, edge, and hybrid infrastructure, and asking what the accumulated evidence says about where this approach genuinely helps and where the open problems remain. We organize the discussion into five parts: the structure of the computing continuum itself and why it resists flat, non-graph representations; the core GNN architectures relevant to this domain, including the temporal and heterogeneous variants that matter once the "graph" in question is changing over time or mixing device types; the case for treating resource allocation as a graph-structured learning problem in the first place; a tour of concrete GNN-based scheduling and allocation systems, from cluster job schedulers to wireless power control to network digital twins; and a set of case studies specific to edge and multi-access scenarios, closing with the open challenges that recur across this literature.

A note on scope before we begin. Much of the most rigorously evaluated work here comes from adjacent but narrower domains, wireless resource allocation, data-center job scheduling, network performance modeling, rather than from systems tested end-to-end on a genuine multi-tier cloud-edge deployment. We think the architectural logic connecting these domains to the cloud-edge continuum is sound and, increasingly, directly demonstrated, but we try to be explicit throughout about which claims come from a paper's own reported numbers and which are our extrapolation from a closely related setting.

2. The Computing Continuum: From Cloud to Edge

Edge computing entered the literature as a response to a fairly concrete complaint about the cloud-only model: pushing every byte of sensor, video, and interaction data to a centralized data center for processing incurs latency, consumes backhaul bandwidth, and creates a single point of both failure and privacy exposure that many applications, industrial control loops, augmented reality, autonomous vehicles, simply cannot tolerate. Shi et al.'s foundational treatment frames edge computing as pushing computation to the network edge, near

where data is generated, and lays out the resulting research agenda around programming models, resource management, and security for this newly distributed setting [5]. Mao et al.'s survey of mobile edge computing from the communication-systems side complements this with a detailed accounting of the joint communication-and-computation resource allocation problem that edge computing creates: a task can be executed locally, offloaded to a nearby edge server, or offloaded further to the cloud, and the right choice depends jointly on wireless channel conditions, queueing delay, energy budget, and the computational demand of the task itself, a genuinely multi-dimensional optimization that the survey catalogs across dozens of proposed formulations [6].

What has changed since those early framings is the recognition that "cloud" and "edge" are not two discrete tiers but points on a continuum, and that a real deployment typically spans several such points simultaneously: a factory floor gateway, a regional telco edge site, and a hyperscale cloud region, each holding a different mix of compute, storage, and proximity to the data source. Gkonis et al.'s survey of IoT-edge-cloud continuum systems catalogs this shift explicitly, describing continuum architectures as systems designed to place and, critically, *re-place* workloads dynamically across tiers as conditions change, rather than pinning a service to a single tier at deployment time, and surveys the orchestration, resource management, and use-case literature that has grown up around this more fluid model [1]. Zhou et al.'s treatment of edge intelligence pushes the framing one step further by asking not just where computation happens but where *learning* happens, arguing that the sheer scale of edge-connected devices, now numbering in the tens of billions and growing faster than the engineering teams that manage them, makes centralized, cloud-only intelligence untenable as a long-term architecture and that intelligence itself, model training and inference alike, needs to be distributed across the same continuum that hosts the data [7].

Why the Continuum Resists Flat Representations

The practical consequence of this multi-tier, dynamically re-placeable model is that any resource allocation decision made on it has to account for a genuinely relational structure: which nodes are physically and logically adjacent, what the current link

latency and bandwidth between them is, which services depend on which other services, and how workload demand is currently distributed across the topology. A flat feature vector, total CPU utilization per node, say, concatenated across all nodes in a fixed order, can encode some of this, but it does so badly: it requires a fixed, pre-specified ordering and count of nodes, it cannot represent the graph's edges (the adjacency and dependency structure) directly, and a model trained on one topology size or shape has no principled way to generalize to a differently sized or shaped one. This is precisely the representational gap that motivated the development of graph neural networks for other domains, molecules, citation networks, social graphs, where the entities of interest are naturally variable in number and connected by meaningful, non-arbitrary edges [4,8]. The cloud-edge continuum, its physical topology, its service dependency graphs, its traffic matrices, exhibits exactly the same structural signature, which is the central reason the literature synthesized in the remainder of this article treats GNNs not as one modeling option among many but as an unusually well-matched tool for this specific class of problem.

3. Graph Neural Network Foundations

Message Passing and Convolutional GNNs

The dominant computational pattern underlying nearly all modern GNN architectures is message passing: each node holds a feature vector (its own attributes, or in a resource-allocation context, something like current load, queue depth, or link capacity), and in each layer of the network, every node aggregates messages from its immediate neighbors, combines them with its own current representation, and produces an updated representation for the next layer. Gilmer et al.'s message-passing neural network (MPNN) framework, developed originally for predicting molecular properties from atomic graphs, formalized this pattern as a general template, explicit message functions, a permutation-invariant aggregation (commonly a sum or mean over neighbors, chosen so the result does not depend on the arbitrary order in which neighbors are enumerated), and an update function, into which most subsequent GNN variants can be slotted as specific choices of those three functions [9]. Kipf and Welling's graph convolutional network (GCN) is, in this framework, a particularly simple and influential instance: it defines the aggregation as a normalized sum

over a node's neighbors (with normalization chosen to account for each node's degree), stacks several such layers, and showed that this scheme substantially improved semi-supervised node classification accuracy on citation-network benchmarks relative to prior graph-regularization approaches, while remaining computationally cheap enough to scale to graphs with many thousands of nodes [2].

Two extensions to this basic scheme matter a great deal for resource-allocation applications specifically. First, Hamilton, Ying, and Leskovec's GraphSAGE addressed a scalability and generalization concern that plain GCN, in its original formulation, did not: GraphSAGE learns a set of aggregation functions rather than a fixed set of per-node embeddings, which means the resulting model can be applied inductively to nodes and even entire graphs it never saw during training, sampling a fixed-size neighborhood around each node rather than requiring access to the full graph at every step [3]. This inductive property is close to a hard requirement for cloud-edge resource allocation, where the number of active servers, edge nodes, and in-flight tasks changes constantly and a model that could only make predictions on the exact graph it was trained on would be useless in production. Second, Veličković et al.'s graph attention network (GAT) replaced GCN's fixed, degree-based neighbor weighting with a learned attention mechanism, letting the model decide, per edge and per layer, how much weight to give each neighbor's message rather than weighting all neighbors by a fixed normalization constant [10]. For resource allocation this matters because not all neighboring relationships are equally informative for a given decision, a heavily loaded neighbor's state may matter far more to a placement decision than a lightly loaded one two hops away, and several of the applied systems discussed later in this article build directly on GAT-style attention for exactly this reason.

Heterogeneous and Temporal Graph Learning

Plain GCN, GraphSAGE, and GAT all assume, at least in their original formulations, a single node type and a single edge type, which is a reasonable simplification for a citation graph but a poor fit for a cloud-edge continuum containing cloud servers, edge gateways, IoT devices, and network links that each behave quite differently and cannot be sensibly averaged together in a single aggregation step. Wang et al.'s heterogeneous

graph attention network (HAN) addressed this by learning attention weights at two levels: node-level attention over neighbors of the same relational type (reachable via the same kind of edge, or "meta-path"), and a separate semantic-level attention over the different relational types themselves, so that the model can learn, for instance, that a service's dependency edges matter more to a given prediction than its co-location edges, without requiring the designer to hand-specify that weighting [11]. This heterogeneous-graph machinery maps naturally onto continuum infrastructure, where cloud nodes, edge nodes, and the workloads placed on them genuinely are different kinds of entities connected by different kinds of relationships, and several of the placement and scheduling systems discussed below adopt heterogeneous or multi-relational graph representations for precisely this reason, even when they do not cite HAN directly.

The second gap plain GCN-style architectures leave open is time: a static graph snapshot captures the topology and load at one instant, but resource allocation in the cloud-edge continuum is, by the article's own title, dynamic, and both the topology (nodes joining and leaving, links appearing and disappearing) and the node/edge attributes (load, queue depth, channel quality) change continuously. Two complementary lines of work address this. Yu, Yin, and Zhu's spatio-temporal graph convolutional network (STGCN), developed for traffic-speed forecasting on road networks, combines graph convolutions that capture spatial dependence between neighboring sensors with temporal convolutions that capture how each sensor's own readings evolve over time, and demonstrated that this joint spatio-temporal treatment produced substantially more accurate short-term forecasts than either a purely temporal (recurrent) or purely spatial (static graph) model alone on real traffic-speed datasets [12]. Rossi et al.'s temporal graph networks (TGN) framework generalizes this further to continuously evolving graphs rather than fixed-interval snapshots, maintaining a per-node memory that is updated as timestamped events, an edge appearing, a node's attributes changing, arrive, and showing that this event-driven memory mechanism outperformed both static GNN baselines and prior discrete-time dynamic graph models on link-prediction and node-classification benchmarks built from evolving interaction graphs [13]. Both threads,

spatio-temporal convolution and continuous-time memory, recur throughout the applied resource-allocation literature discussed later, because a cloud-edge continuum's load and topology are exactly the kind of continuously evolving graph signal these architectures were built to consume.

4. GNN Architectures for Scheduling and Resource Allocation

Framing resource allocation as a graph-structured decision problem is only useful if it can be turned into a trainable policy, and the literature has converged on combining GNN encoders with reinforcement learning (RL) decision layers far more often than with supervised learning alone, precisely because allocation decisions are sequential (each decision changes the state the next decision is made from) and the "correct" allocation is rarely available as a labeled example the way, say, an image's class label is. Mao et al.'s DeepRM was an early instance of this general idea, though without a graph encoder: it represented a cluster scheduling problem as an image-like grid of resource occupancy and used a policy-gradient RL agent to learn packing and scheduling decisions directly from experience, demonstrating that a learned policy could match or exceed hand-tuned heuristics on job-packing objectives without being told the scheduling rules explicitly [14]. DeepRM's grid representation, however, inherited the same fixed-size, non-relational limitation described above, and the natural next step, taken by the same research group a few years later, was to replace the grid with an actual graph.

Mao et al.'s Decima system is the clearest and most thoroughly evaluated instance of this evolution. Decima represents each incoming job as a DAG of stages with data-dependency edges, embeds this DAG with a graph neural network to produce a fixed-size summary that is invariant to the number and arrangement of stages, and combines that embedding with a policy network trained via reinforcement learning to decide which stage to schedule next and how many executors to allocate to it [15]. Critically, the paper reports concrete, verifiable performance numbers rather than only qualitative claims: relative to hand-tuned scheduling heuristics, Decima reduces average job completion time on TPC-H query workloads by at least 21%, with the improvement growing to as much as a 2× reduction in average job completion time during periods of high cluster load, and

by 32-43% relative to prior learned or heuristic multi-resource schedulers such as Graphene when scheduling jointly over CPU and memory [15]. These numbers matter beyond their specific values because they demonstrate something the graph-representation argument alone cannot: that the GNN encoder's chief practical benefit, generalizing a single learned policy across DAGs of varying size and shape without retraining, translates into a measurable scheduling-quality improvement on a workload the model was not specifically tuned for, and that the improvement is largest precisely when the scheduling problem is hardest, under high load, which is exactly the regime where a cloud-edge continuum's dynamic, bursty demand is most likely to matter.

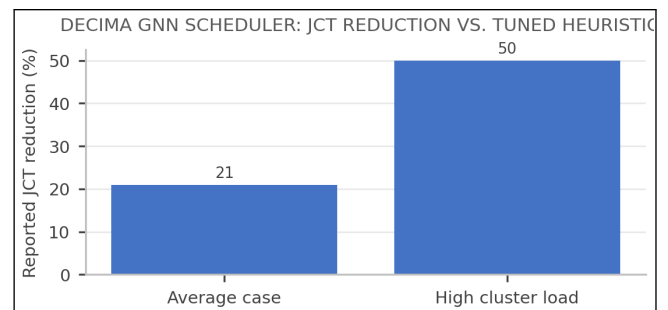


Figure 1: Reduction in average job completion time (JCT) that Decima's graph-neural-network scheduler achieves over hand-tuned heuristics on TPC-H workloads -- at least 21% on average, growing to a reduction equivalent to a 2x speedup during periods of high cluster load, exactly when scheduling is hardest [15].

A parallel and largely independently developed line of GNN-based resource allocation work targets wireless power control and spectrum allocation rather than cluster job scheduling, but the underlying logic is close to identical: represent interfering transmitter-receiver links as nodes and edges of an interference graph, and learn a GNN policy that maps this graph to a power or resource allocation decision. Eisen and Ribeiro's random edge graph neural network (REGNN) formulates power control in this way, arguing, and empirically supporting the claim, that because the interference graph's structure (which links interfere with which) is the primary driver of the optimal allocation, a GNN policy conditioned on that structure achieves strong performance relative to model-based heuristic benchmarks while using a number of learnable parameters that does not grow with the size of the network, and that policies trained on one network size transfer to differently sized networks without retraining [16]. Shen, Zhang, Song, and Letaief's survey of GNNs for

wireless communications situates REGNN within a broader family of GNN-based resource allocation methods for power control, beamforming, and user scheduling, and makes a specific theoretical argument for why the approach works as well as it empirically does: many classical wireless resource allocation problems possess a permutation-equivariance property (relabeling the users or links in a network should relabel the optimal solution correspondingly, without otherwise changing it), which is precisely the symmetry that GNNs are mathematically constructed to respect, giving the architecture a genuine inductive-bias advantage over generic fully connected networks on this class of problem rather than a merely empirical one [17].

A third strand applies GNNs not to the allocation decision itself but to modeling the network whose state that decision depends on, an increasingly important distinction as decisions are made more frequently and the cost of querying a real network or running a packet-level simulator to evaluate a candidate allocation becomes prohibitive. Rusek et al.'s RouteNet represents a network's topology, routing, and traffic matrix as a graph and trains a GNN to predict per-flow performance metrics, delay, jitter, and loss, directly from that representation, reporting that the resulting model generalizes to network topologies, routing configurations, and traffic intensities well outside its training distribution far more reliably than prior neural approaches that did not encode topology explicitly [18]. Ferriol-Galmés et al.'s RouteNet-Fermi extends this to a considerably richer traffic and queueing model and reports concrete accuracy figures that make the approach's practical value legible: a mean relative delay error of 6.24% against packet-level simulation on topologies of up to 300 nodes, degrading only modestly, to roughly 5.6% and 11% respectively, when evaluated against real packet traces and a physical hardware testbed rather than simulation alone, with corresponding jitter and packet-loss errors reported in the same range across several distinct traffic models [19]. A model that can predict network performance this accurately, and roughly four to five orders of magnitude faster than running an actual packet-level simulator, functions as a differentiable or at least rapidly queryable surrogate that a resource allocation policy, learned or heuristic, can consult to evaluate candidate allocations before committing to one, which is precisely the workflow several of the case-study systems in the next

section build on.

5. Case Studies: GNN-Based Systems for the Cloud-Edge Continuum

Moving from the general architectural building blocks to systems built specifically for edge and multi-tier cloud-edge scenarios, several recent papers apply the GNN-plus-RL pattern established by Decima and REGNN directly to task offloading, virtual network function (VNF) placement, and microservice resource prediction, the three problem classes that dominate the applied cloud-edge resource allocation literature. Yang et al.'s RLCS targets mobile edge computing task offloading and resource scheduling jointly, using a graph representation of the edge-cloud topology and available resources as input to a reinforcement learning agent, and reports that the resulting system outperforms both heuristic and prior reinforcement-learning baselines across multiple experimental settings, with the paper's stated design goal being specifically to improve robustness to the kind of variable connectivity and resource availability that characterizes real mobile edge deployments rather than the cleaner, more static settings typical of cluster scheduling benchmarks [20].

Table 1: Representative GNN-based systems for resource allocation and network modeling across the cloud-edge continuum, with the effect or accuracy figure each cited paper reports. [15,19,16,21,22,23].

System	Target Problem	GNN Role	Reported Effect
Decima [15]	Cluster job/DAG scheduling	Encodes job DAGs and cluster state for an RL scheduling policy	At least 21% lower avg. job completion time vs. tuned heuristics; up to 2x under high load
RouteNet-Fermi [19]	Network performance modeling (digital twin)	Predicts per-flow delay/jitter/loss from topology + traffic graph	6.24% mean relative delay error vs. packet-level simulation (up to 300-node topologies)
REGNN [16]	Wireless power control / spectrum allocation	Maps interference graph to per-link power allocation	Near-benchmark power-control performance with parameter count independent of network size
GNN-DR L VNF placement [21]	VNF forwarding-graph placement (SDN/NFV-IoT)	Encodes joint physical-network + VNF-graph state for a placement policy	Joint optimization of acceptance ratio and resource cost under latency constraints
Two-stage GCN-DRL [22]	Service function chain embedding, multi-datacenter	GCN encoder produces datacenter/node embeddings for a separate RL placement stage	Decouples graph encoding from placement decisions to scale to more candidate hosts

System	Target Problem	GNN Role	Reported Effect
GraphGRU [23]	Per-microservice resource demand forecasting	Graph encodes service-dependency structure; GRU captures per-service temporal history	Combines dependency-aware graph structure with temporal load history for demand prediction

Virtual network function placement and service function chain (SFC) embedding, the problem of deciding which physical or virtual machines should host which pieces of a chained network service, subject to resource and latency constraints, has independently converged on the same GNN-plus-RL pattern. Xie et al. formulate VNF forwarding-graph placement in SDN- and NFV-enabled IoT networks as a graph neural network assisted deep reinforcement learning problem, using a GNN to encode the joint state of the physical network topology and the VNF forwarding graph being placed onto it, and using the resulting representation to drive a placement policy that must jointly satisfy per-VNF resource constraints and end-to-end latency budgets [21]. Xiao et al. push a closely related formulation to a larger scale, multi-datacenter SFC embedding, using a two-stage architecture in which a graph convolutional encoder first produces datacenter-level and node-level embeddings and a subsequent reinforcement-learning stage uses those embeddings to make chain-placement decisions across the multi-datacenter topology, explicitly targeting the scalability limitations that make single-stage placement policies expensive to train and evaluate as the number of candidate hosts grows [22]. Considered together, these two systems are a useful illustration of a recurring design choice in this literature: rather than asking a single GNN-RL policy to reason jointly over an entire multi-tier topology at once, several systems factor the problem into a graph-encoding stage that produces a compact, size-invariant summary and a separate decision stage that consumes that summary, a division of labor that keeps the RL agent's action space and per-step decision cost from scaling directly with the size of the underlying infrastructure graph.

Not every application in this space is a placement or scheduling decision in the classical sense; some are prediction problems that feed into an allocation decision made by a separate controller. He, Su, and Ye's GraphGRU targets exactly this: predicting near-future resource demand (CPU and memory utilization) for individual microservices in a containerized cluster, representing the cluster's service-dependency structure

as a graph and combining a graph neural network with a gated recurrent unit to capture how a microservice's resource demand is shaped both by its own recent history and by the current load on the services it depends on or is called by [23]. This kind of demand-forecasting model is a natural companion to the placement systems described above: a placement or autoscaling controller needs an estimate of near-future demand to make a good decision, and GraphGRU's explicit modeling of inter-service dependency, rather than treating each microservice's load as an independent time series, is a direct application of the same relational-structure argument this article has made throughout, applied to forecasting rather than to the allocation decision itself.

Finally, Qin et al.'s work on dynamic task offloading and resource allocation in vehicular edge computing networks illustrates how directly this line of research has converged on the article's own framing: the paper explicitly targets *dynamic* resource allocation, using a graph attention network to encode the time-varying topology of vehicles, roadside units, and edge servers (a setting where connectivity genuinely changes on a timescale of seconds as vehicles move), combined with a deep reinforcement learning offloading policy trained on top of the resulting graph-attention representation [24]. That a graph-attention-based formulation, rather than a plain GCN or a non-graph baseline, was the chosen architecture here is consistent with the broader pattern in this section: as the target environment's dynamism and heterogeneity increase, from cluster scheduling's relatively stable DAGs, through SFC placement's moderately dynamic multi-datacenter topologies, to vehicular edge computing's rapidly changing connectivity, the systems built for it increasingly favor GAT-style learned attention over GCN's fixed neighbor weighting, for exactly the reason Veličković et al. originally motivated attention in the first place: not every neighbor's current state deserves equal weight, and which neighbors matter most is itself changing over time [10].

6. Open Challenges

Several gaps recur across the literature synthesized here and are worth stating plainly. First, evaluation realism is uneven: Decima and REGNN report rigorous, quantified comparisons against strong baselines [15,16], and RouteNet-Fermi validates against a physical testbed [19],

but several edge- and VNF-placement-specific systems report only relative outperformance without the same reproducible detail, making it harder to translate a claimed improvement into an expectation for one's own deployment. Second, almost none of the systems surveyed report results on infrastructure spanning all three tiers of the continuum simultaneously under realistic, time-varying inter-tier network conditions; most target a single tier or a two-tier edge-cloud split, leaving cross-tier dynamics comparatively under-explored even where the architectural ideas would plausibly extend.

Third, GNN inference scalability becomes a genuine constraint as the continuum grows: a forward pass typically aggregates over several hops of neighbors, and while GraphSAGE was specifically designed to keep this tractable via neighbor sampling [3], most resource-allocation systems built on this machinery still assume the inference step itself can run on capable hardware, an assumption more comfortable for a cloud-side scheduler like Decima than for a decision made on or near a constrained edge device. Fourth, most systems reviewed here treat the graph as static within an episode or, at best, spatio-temporally smooth in the STGCN sense [12], rather than adopting the continuous-time, event-driven update model TGN-style architectures offer [13]; since topology change in the continuum, a device joining, a link failing, a container migrating, is itself an event stream, this is a meaningful mismatch between available tools and how most applied systems currently use them.

Fifth, heterogeneity across node and edge types is handled inconsistently: HAN-style heterogeneous attention offers a principled mechanism for combining cloud nodes, edge nodes, and multiple relation types [11], yet the applied systems in the case-studies section largely use simpler, homogeneous graph representations, leaving real structural differences unmodeled. Sixth, training stability and sample efficiency for the reinforcement-learning component remain a practical bottleneck, and the field has not converged on standardized benchmarks that would let a new method be compared against prior work as cleanly as, say, image classification methods can be. None of these gaps strike us as reasons to doubt the underlying premise, but they are the honest state of the evidence.

7. Conclusion

The literature reviewed here supports a fairly specific and, we think, well-evidenced claim: dynamic resource allocation across the cloud-edge continuum, spanning cluster job scheduling, wireless power control, network performance modeling, and edge task offloading, is a problem whose natural representation is a graph, and graph neural networks, precisely because they are built to consume variable-size, relationally structured input and generalize across it, are a genuinely well-matched tool for learning allocation policies on that representation rather than a fashionable relabeling of existing techniques. The evidence for this claim is strongest where it has been measured most carefully: Decima's double-digit percentage reduction in job completion time over hand-tuned heuristics, growing further under high load [15], REGNN's near-model-based performance with a parameter count that does not scale with network size [16], and RouteNet-Fermi's single-digit-percentage error against both simulation and physical hardware [19] are not marginal results; they represent a consistent pattern of a structurally motivated architectural choice translating into measurable, reproducible gains.

At the same time, the path from these individually strong results to a genuinely deployed, full-continuum resource allocation system remains only partially walked. The case-study systems reviewed here, spanning mobile edge task offloading, VNF and service-function-chain placement, microservice demand forecasting, and vehicular edge computing, demonstrate that the underlying GNN-plus-RL pattern transfers successfully across a wide range of concrete problem formulations [20,21,22,23,24], which is itself meaningful evidence of the pattern's robustness rather than its being a one-off fit to a single benchmark. What remains comparatively underdeveloped is evaluation that spans the full, heterogeneous, continuously evolving cloud-edge topology at once, that uses the heterogeneous and temporal graph-learning tools the foundational GNN literature has already developed [11,13] rather than simpler static or homogeneous approximations, and that reports the kind of rigorous, reproducible quantitative comparison that the strongest papers in this synthesis already model for the rest of the field. Closing that gap, we think, is less a matter of inventing new architectural ideas than of the systems and networking community doing the harder, less glamorous work of building and evaluating on infrastructure that actually

looks like the messy, multi-tier continuum this article has described throughout.

8. References

- Gkonis, P., Giannopoulos, A., Trakadas, P., Masip-Bruin, X., D'Andria, F. (2023) A Survey on IoT-Edge-Cloud Continuum Systems: Status, Challenges, Use Cases, and Open Issues. *Future Internet*, 15(12): 383. DOI: [10.3390/fi15120383](https://doi.org/10.3390/fi15120383)
- Kipf, T.N., Welling, M. (2017) Semi-Supervised Classification with Graph Convolutional Networks. *International Conference on Learning Representations (ICLR)*. DOI: [10.48550/arXiv.1609.02907](https://doi.org/10.48550/arXiv.1609.02907)
- Hamilton, W.L., Ying, R., Leskovec, J. (2017) Inductive Representation Learning on Large Graphs. *Advances in Neural Information Processing Systems (NeurIPS)* 30. DOI: [10.48550/arXiv.1706.02216](https://doi.org/10.48550/arXiv.1706.02216)
- Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., Yu, P.S. (2021) A Comprehensive Survey on Graph Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1): 4-24. DOI: [10.1109/TNNLS.2020.2978386](https://doi.org/10.1109/TNNLS.2020.2978386)
- Shi, W., Cao, J., Zhang, Q., Li, Y., Xu, L. (2016) Edge Computing: Vision and Challenges. *IEEE Internet of Things Journal*, 3(5): 637-646. DOI: [10.1109/JIOT.2016.2579198](https://doi.org/10.1109/JIOT.2016.2579198)
- Mao, Y., You, C., Zhang, J., Huang, K., Letaief, K.B. (2017) A Survey on Mobile Edge Computing: The Communication Perspective. *IEEE Communications Surveys & Tutorials*, 19(4): 2322-2358. DOI: [10.1109/COMST.2017.2745201](https://doi.org/10.1109/COMST.2017.2745201)
- Zhou, Z., Chen, X., Li, E., Zeng, L., Luo, K., Zhang, J. (2019) Edge Intelligence: Paving the Last Mile of Artificial Intelligence With Edge Computing. *Proceedings of the IEEE*, 107(8): 1738-1762. DOI: [10.1109/JPROC.2019.2918951](https://doi.org/10.1109/JPROC.2019.2918951)
- Zhou, J., Cui, G., Hu, S., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C., Sun, M. (2020) Graph Neural Networks: A Review of Methods and Applications. *AI Open*, 1: 57-81. DOI: [10.1016/j.aiopen.2021.01.001](https://doi.org/10.1016/j.aiopen.2021.01.001)
- Gilmer, J., Schoenholz, S.S., Riley, P.F., Vinyals, O., Dahl, G.E. (2017) Neural Message Passing for Quantum Chemistry. *Proceedings of the 34th International Conference on Machine Learning (ICML)*, PMLR 70: 1263-1272. DOI: [10.48550/arXiv.1704.01212](https://doi.org/10.48550/arXiv.1704.01212)
- Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., Bengio, Y. (2018) Graph Attention Networks. *International Conference on Learning Representations (ICLR)*. DOI: [10.48550/arXiv.1710.10903](https://doi.org/10.48550/arXiv.1710.10903)
- Wang, X., Ji, H., Shi, C., Wang, B., Ye, Y., Cui, P., Yu, P.S. (2019) Heterogeneous Graph Attention Network. *Proceedings of the World Wide Web Conference (WWW)*, 2022-2032. DOI: [10.1145/3308558.3313562](https://doi.org/10.1145/3308558.3313562)
- Yu, B., Yin, H., Zhu, Z. (2018) Spatio-Temporal Graph Convolutional Networks: A Deep Learning Framework for Traffic Forecasting. *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI)*, 3634-3640. DOI: [10.48550/arXiv.1709.04875](https://doi.org/10.48550/arXiv.1709.04875)
- Rossi, E., Chamberlain, B., Frasca, F., Eynard, D., Monti, F., Bronstein, M. (2020) Temporal Graph Networks for Deep Learning on Dynamic Graphs. *ICML Workshop on Graph Representation Learning and Beyond*. DOI: [10.48550/arXiv.2006.10637](https://doi.org/10.48550/arXiv.2006.10637)
- Mao, H., Alizadeh, M., Menache, I., Kandula, S. (2016) Resource Management with Deep Reinforcement Learning. *Proceedings of the 15th ACM Workshop on Hot Topics in Networks (HotNets)*. DOI: [10.1145/3005745.3005750](https://doi.org/10.1145/3005745.3005750)
- Mao, H., Schwarzkopf, M., Venkatakrishnan, S.B., Meng, Z., Alizadeh, M. (2019) Learning Scheduling Algorithms for Data Processing Clusters. *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM)*, 270-288. DOI: [10.1145/3341302.3342080](https://doi.org/10.1145/3341302.3342080)
- Eisen, M., Ribeiro, A. (2020) Optimal Wireless Resource Allocation with Random Edge Graph Neural Networks. *IEEE Transactions on Signal Processing*, 68: 2977-2991. DOI: [10.1109/TSP.2020.2988255](https://doi.org/10.1109/TSP.2020.2988255)
- Shen, Y., Zhang, J., Song, S.H., Letaief, K.B. (2023) Graph Neural Networks for Wireless Communications: From Theory to Practice. *IEEE Transactions on Wireless Communications*, 22(5): 3554-3569. DOI: [10.1109/TWC.2022.3219840](https://doi.org/10.1109/TWC.2022.3219840)
- Rusek, K., Suárez-Varela, J., Almasan, P., Barlet-Ros, P., Cabellos-Aparicio, A. (2020) RouteNet: Leveraging Graph Neural Networks for Network Modeling and Optimization in SDN. *arXiv preprint*. DOI: [10.48550/arXiv.1910.01508](https://doi.org/10.48550/arXiv.1910.01508)
- Ferriol-Galmés, M., Paillisse, J., Suárez-Varela, J., Rusek, K., Xiao, S., Shi, X., Cheng, X., Barlet-Ros, P., Cabellos-Aparicio, A. (2023) RouteNet-Fermi: Network Modeling With Graph Neural Networks. *IEEE/ACM Transactions on Networking*, 31(6): 3080-3095. DOI: [10.1109/TNET.2023.3269983](https://doi.org/10.1109/TNET.2023.3269983)
- Yang, S., Zhang, L., Cui, L., Dong, Q., Xiao, W., Luo, C. (2023) RLCS: Towards a Robust and Efficient Mobile Edge Computing Resource Scheduling and Task Offloading System Based on Graph Neural Network. *Computer Communications*, 206: 38-50. DOI: [10.1016/j.comcom.2023.04.020](https://doi.org/10.1016/j.comcom.2023.04.020)
- Xie, Y., Huang, L., Kong, Y., Wang, S., Xu, S., Wang, X., Ren, J. (2022) Virtualized Network Function Forwarding Graph Placing in SDN and NFV-Enabled IoT Networks: A Graph Neural Network Assisted Deep Reinforcement Learning Method. *IEEE Transactions on Network and Service Management*, 19(1): 524-537. DOI: [10.1109/TNSM.2022.3145201](https://doi.org/10.1109/TNSM.2022.3145201)

[10.1109/TNSM.2021.3123460](https://doi.org/10.1109/TNSM.2021.3123460)

22. Xiao, D., Zhang, J.A., Liu, X., Qu, Y., Ni, W., Liu, R.P. (2023) A Two-Stage GCN-Based Deep Reinforcement Learning Framework for SFC Embedding in Multi-Datacenter Networks. *IEEE Transactions on Network and Service Management*, 20(4): 4297-4312. DOI: [10.1109/TNSM.2023.3284293](https://doi.org/10.1109/TNSM.2023.3284293)
23. He, H., Su, L., Ye, K. (2022) GraphGRU: A Graph Neural Network Model for Resource Prediction in Microservice Cluster. 2022 IEEE 28th International Conference on Parallel and Distributed Systems (ICPADS). DOI: [10.1109/ICPADS56603.2022.00071](https://doi.org/10.1109/ICPADS56603.2022.00071)
24. Qin, B., He, A., Pan, H., Si, X., Dai, Y., Huang, X., Zhang, Y. (2024) Dynamic Task Offloading and Resource Allocation in Vehicle Edge Computing and Networks: A Graph Attention-Based Deep Reinforcement Learning Approach. 2024 IEEE International Conference on High Performance Computing and Communications (HPCC). DOI: [10.1109/HPCC64274.2024.00158](https://doi.org/10.1109/HPCC64274.2024.00158)

9. Declarations

Funding

This literature synthesis was conducted without support from any specific grant, funding agency, or commercial sponsor.

Conflict of Interest

The authors have no conflicts of interest to disclose.

Data Availability Statement

This article is a literature synthesis and did not generate new data; all quantitative figures discussed are drawn from, and attributed to, the cited primary sources, which readers should consult for underlying data and code.

Ethical Approval

Not applicable. No human participants, animal subjects, or primary data collection were involved in this literature synthesis.