

Autonomic GNN-Based Shortest-Path Routing in LEO Networks via Imitation and Transfer Learning

Federica Filippini*, Fabio Villa*, Michele Ciavotta*, Marco Savi*

*University of Milano-Bicocca, Milan, Italy

Email: {federica.filippini, fabio.villa, michele.ciavotta, marco.savi}@unimib.it

Abstract—Low Earth Orbit (LEO) satellite networks represent a key enabler for global connectivity, yet their highly dynamic topologies severely challenge classical routing algorithms that require static tables or frequent global recomputation. To enable continuous self-adaptation, we propose an autonomic, data-driven routing framework based on Graph Neural Networks (GNNs). Adopting an imitation learning approach, we formulate decentralized shortest-path routing as a supervised edge classification task: given a network graph, the model autonomously infers whether each inter-satellite link belongs to the optimal path. To overcome data scarcity and ensure resilient generalization across changing topologies, we introduce a transfer learning pipeline that pre-trains on synthetic spatial graphs before fine-tuning on real Iridium NEXT topologies. Furthermore, replacing 2D geodetic coordinates with a 3D Earth-Centered, Earth-Fixed (ECEF) representation eliminates wrap-around artifacts, substantially improving spatial reasoning. The best model identifies the shortest path in 97.1% of cases, while overall identifying a valid source-to-destination routing in 99.0% on unseen test sets, successfully enabling self-organizing routing.

Index Terms—Graph Neural Networks, LEO satellite networks, shortest-path routing, edge classification, imitation learning, transfer learning, Iridium NEXT.

I. INTRODUCTION

Over the last decades, technological progress has led to an exponential growth in the amount of data generated worldwide, placing significant pressure on traditional terrestrial network infrastructures. In many rural, remote, or sparsely populated regions, deploying wired or cellular networks remains economically and technically challenging [1]. Consequently, Low Earth Orbit (LEO) satellite constellations have emerged as a key complement to terrestrial infrastructures, offering global coverage, low latency (around 10ms), and robust support for 5G and emerging 6G services [2], [3]. This technological shift has triggered massive investments, with the satellite Internet market projected to grow from 9.53 to 33.4 billion USD by 2034 [4].

Unlike traditional geostationary systems, LEO constellations achieve these properties through highly dynamic topologies. Satellites orbit continuously, changing their relative positions and Inter-Satellite Link (ISL) availability every few minutes. Modern architectures leverage ISLs to reduce reliance on ground stations, forming self-organizing spatial mesh networks capable of autonomous, end-to-end routing in space [2]. However, this structural fluidity introduces fundamental self-management routing challenges, which are not addressed by classical algorithms and protocols such as Dijkstra’s and Open Shortest Path First. First, they assume static or slowly changing topologies,

whereas LEO networks require continuous self-adaptation to link activations and deactivations, particularly around the polar seam. Second, traditional approaches demand frequent global recomputations or massive routing tables, violating the strict computational and memory constraints of LEO satellites. Third, limited ground-to-space link budgets make frequent topology broadcasts too expensive for practical applications [5], necessitating decentralized, autonomic routing solutions.

Graph Neural Networks (GNNs) have recently emerged as a promising approach to this class of problems [6], [7]. A satellite constellation naturally maps to a graph where nodes are satellites and edges are ISLs. GNNs operate directly on these graph-structured inputs; their permutation equivariance and shared parameters make them topology-agnostic and scalable, which is highly advantageous in self-organizing domains that continuously change configuration. Although recent work has explored GNNs for routing in terrestrial [8], [9] and satellite [10], [11] networks, prior approaches often solve a regression problem (predicting link distance) or rely on centralized Software-Defined Networking (SDN) controllers, which introduce single points of failure and latency bottlenecks that conflict with the autonomic computing goal of decentralized self-management.

In this paper, we propose a decentralized, autonomic GNN routing framework for the Iridium NEXT LEO constellation [12]. Instead of relying on static routing tables or repeated shortest-path computations, we propose an imitation learning paradigm, where the GNN model learns from expert demonstrations to autonomously infer path membership directly from the network topology and link-level features, enabling each satellite to make fast, self-adaptive forwarding decisions. Specifically:

- We frame shortest-path routing as a supervised edge classification task via imitation learning [13]. The GNN learns from expert demonstrations (generated via Dijkstra’s algorithm) to autonomously predict optimal paths between ground stations, enabling decentralized decision-making
- To overcome data scarcity and ensure robust self-adaptation across unseen topologies, we propose a data augmentation strategy and design a transfer learning pipeline. Pre-training on diverse synthetic spatial graphs before fine-tuning on real Iridium NEXT data yields a 15% performance improvement in under 200 GPU-seconds.
- We demonstrate that replacing traditional 2D geodetic coordinates (latitude/longitude) node features with 3D Earth-Centered, Earth-Fixed (ECEF) coordinates eliminates wrap-around artifacts, improving the model’s spatial

reasoning and reconstruction performance. This yields around 13% more graphs where all edges are classified correctly, with the best model identifying the shortest path in 97.1% of cases, while overall identifying a valid source-to-destination routing in 99% on unseen test sets.

By integrating machine learning for network management, geometric deep learning, and transfer learning, we deliver a unified framework that advances autonomic routing and self-organization in dynamic, resource-constrained LEO networks.

The remainder of the paper is organized as follows. Section II provides background on the Iridium NEXT constellation and GNNs. Section III reviews the related work. Section IV defines the system model and problem formulation. Section V details our data augmentation and transfer learning pipeline. Sections VI and VII present the experimental analysis. Section VIII discusses key findings, and Section IX concludes the paper.

II. BACKGROUND

This section describes the technologies and models considered in this work, in particular the Iridium NEXT satellite topology (Section II-A) and the GNN models (Section II-B), with a particular focus on GNN architecture variants (Section II-C).

A. The Iridium NEXT Constellation

Iridium NEXT is the second generation of the Iridium LEO satellite system, consisting of 66 active satellites arranged in 6 orbital planes of 11 satellites each [12], [14]. Satellites orbit at an altitude of approximately 780 km in near-polar orbits with an inclination of 86.4° , completing one orbit every 100 minutes and covering a footprint radius of 2209 km. Orbital planes are separated by 31.6° at the equator, while an additional 22° separates the two reverse rotation planes (planes 6 and 1) creating the “seam”, a transition zone between ascending and descending trajectories. The constellation provides continuous global coverage thanks to the Earth’s rotation.

As shown in Fig. 1, each satellite is equipped with four ISLs: two connecting adjacent satellites within the same orbital plane (*intra-plane* links, which are always active and have nearly-constant distance) and two connecting satellites in neighboring planes (*inter-plane* links, whose distance varies with orbital phase). Across the seam, reversed orbital directions prevent stable cross-links, causing periodic topology changes roughly every 10 minutes. This mesh structure enables multi-hop routing entirely in space, reducing round-trips to ground stations to a single uplink (source ground station to satellite) and downlink (satellite to destination ground station)¹. The resulting topology is a dynamic graph $G_t = (V, E_t)$ whose edge set E_t changes continuously with satellite motion, dictating the need for efficient adaptive routing. However, it shows a structured regularity that distinguishes it from generic spatial graphs: the degree distribution is highly predictable, with most satellites having exactly 4 neighbors and those near the polar seam temporarily having 2 (inter-plane ISLs periodically deactivate

¹These links are not ISLs but we model them identically in our problem as directed edges in the graph.



Fig. 1: The Iridium NEXT constellation: 66 satellites in 6 polar orbital planes forming a continuous spatial mesh [15].

as satellites pass through the polar seam). The graph diameter (longest shortest path in hops) ranges from 5 to approximately 12, with a median around 7, depending on the relative positions of the active ground stations.

B. Graph Neural Networks

GNNs [16] represent a class of deep learning architectures designed to process graph-structured data preserving the relational dependencies and inherent structural information. Given a graph $G = (V, E)$ with initial node features $\{h_u^{(0)} = x_u\}_{u \in V}$, each node u updates its hidden representation through an iterative *message-passing* mechanism, aggregating information from its neighborhood $N(u)$, as shown in Fig. 2a, by computing:

$$h_u^{(k+1)} = \text{UPDATE}^{(k)}\left(h_u^{(k)}, \text{AGGREGATE}^{(k)}(\{h_v^{(k)}\}_{v \in N(u)})\right). \quad (1)$$

The AGGREGATE function must be permutation invariant to ensure the model remains equivariant to the ordering of neighbors [16], while the UPDATE function produces a reformulated latent representation. The depth of a GNN, defined by the number of message-passing layers (or number of iterations) K , determines its receptive field: the final embedding $z_u = h_u^{(K)}$ encodes both local features and information from the K -hop neighborhood of u . This allows the model to capture increasingly distant structural patterns and feature correlations.

A known limitation is *over-smoothing*: as K increases, node representations converge toward a common value, losing discriminative power [16] (see Fig. 2b). To mitigate this, modern architectures employ residual connections, normalization layers, or hybrid strategies to maintain representational diversity. In this work, we adopt a residual concatenation strategy (later defined in Algorithm 1) that directly counteracts this effect by anchoring each iteration to the original encoder output.

C. GNN Variants and Architecture Choices

Several GNN architectures have been proposed for graph-structured prediction tasks. Graph Convolutional Networks (GCNs) [16] apply a shared linear transformation followed by neighborhood averaging, which is computationally efficient but limits expressiveness. Message Passing NNs (MPNNs) [16] generalize GCNs by allowing arbitrary learned aggregation

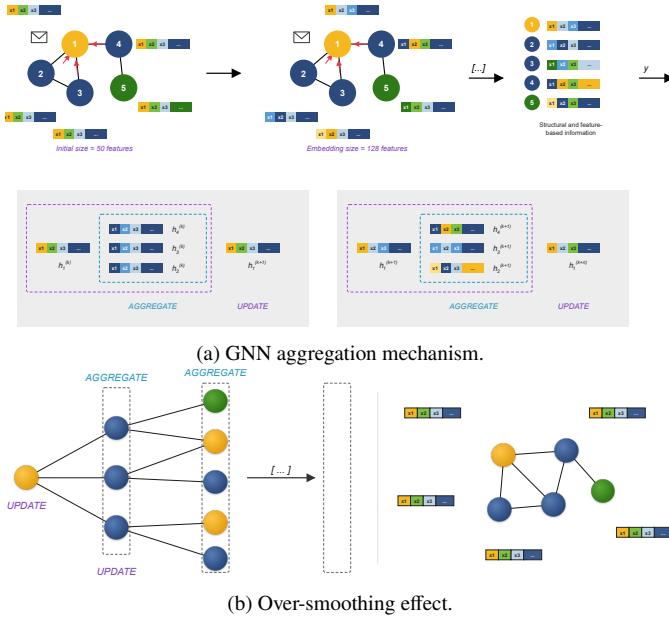


Fig. 2: Neighborhood aggregation mechanism and over-smoothing effect in GNNs [16].

functions, which is the paradigm adopted in this work. Graph Attention Networks (GATs) [17] further weight neighborhood messages by learned attention coefficients; this reduces the influence of irrelevant neighbors, at the price of an additional computational overhead that may be unnecessary when the neighborhood is already geometrically constrained, as in a satellite mesh where each node has at most four physical links.

In this work, we adopt the *Graph Networks (GN)* framework [7], which extends message passing to include explicit edge and global-graph representations jointly updated at each step. Concretely, a GN maintains three latent representations: $\mathbf{h}_e$ for each edge, $\mathbf{h}_v$ for each node, and $\mathbf{g}$ for the graph as a whole. These are updated sequentially: edges first (by aggregating sender/receiver node features), then nodes (by aggregating incoming edge messages), and finally the global attribute (by pooling all node and edge summaries). The choice is motivated by three factors. First, the explicit edge representations allow the model to directly classify edges as path members, without projecting edge information through node aggregation. Second, the global attribute provides a natural representation for graph-level context (e.g., source and destination embedding) that is broadcast to all nodes and edges, ensuring that routing-intent information is globally visible without requiring additional positional encodings. Third, GN adopts a block-compositional design featuring an Encoder, Core, and Decoder that share the same code path and can be individually replaced or retrained, which facilitates our transfer learning pipeline (see Section V).

III. RELATED WORK

Recent literature has increasingly explored GNNs for routing optimization across diverse network domains, including terrestrial infrastructure, SDNs, and satellite constellations. However, existing approaches often struggle to balance the need for

decentralized execution with the highly dynamic, resource-constrained nature of LEO networks. In the following, we review literature approaches focused on GNN-based routing in LEO [10], [11] and terrestrial [8], [9], [18] networks, on SDNs [19], and on the increasingly-adopted Deep Reinforcement Learning (DRL) approaches [20]. Table I summarizes the reviewed works. Overall, our approach differs by (i) treating routing as edge classification rather than regression or next-hop prediction, (ii) operating on the full graph with a global relational architecture, and (iii) supporting decentralized inference through a split ground-satellite execution model.

In LEO networks, [10] proposes GLR, a GNN-based method for on-board routing in small satellite networks, separating low-order and high-order feature extraction and solving a *regression* problem to predict communication distance. While it demonstrates near-optimal performance with low complexity, it limits each satellite's view to a fixed neighborhood and does not exploit a global graph-to-graph architecture. Alternatively [11] integrates a GNN with an SDN controller to manage link reliability and seamless handovers in LEO networks. The GNN provides a global network state to the centralized controller, which then selects routing paths. However, this centralized SDN architecture introduces a single point of failure and is difficult to decentralize to individual satellites.

In terrestrial contexts, [8] presents Grafnet, a table-less IP router combining a GNN, an ANN, and an attention mechanism to output per-port forwarding probabilities. Grafnet achieves up to 98% accuracy and microsecond inference, but operates on *static* topologies where the primary challenge is IP-address mapping rather than dynamic topology adaptation. Similarly, [9] trains a Gated GNN with edge attention to generate distributed routing protocols for fixed terrestrial networks, and [18] applies a GNN to abstract shortest-path problems using a greedy per-hop strategy. Crucially, these approaches operate on static or abstract graphs without spatial coordinates, failing to address the rapid structural changes characteristic of LEO constellations.

Intelligent SDN routing is addressed in [19] by introducing a graph-aware convolution structure. The deep learning model runs on a controller to return optimal paths; although highly accurate, its centralized architecture and reliance on static topologies make it poorly suited to our LEO domain.

Exploring dynamic adaptability, [20] proposes G-Routing, which combines a GNN with DRL to enable flexible online routing. While it shows fast convergence for multi-metric QoS optimization, its reliance on reinforcement signals requires online interaction with the environment, making it harder to pre-train offline and deploy on resource-constrained satellites.

As a foundational basis for our work, the GN framework [7] unifies message passing and graph updates into a single paradigm, proposing the Encoder-Core-Decoder architecture adopted in this work.

IV. PROBLEM FORMULATION AND SYSTEM MODEL

This section outlines the proposed framework for data-driven routing in LEO satellite networks. Section IV-A formalizes how a LEO constellation is modeled as a spatial graph, highlighting

TABLE I: Comparison of GNN-based routing approaches.

Ref.	Domain	Task	Arch.	Key difference
[10]	LEO	Regression	Distr. GNN+FC	Local view only
[11]	LEO	State est.	GNN+SDN	Centralized; no path output
[8]	IP	Port classif.	GNN+ANN	Static topology
[9]	IP	Protocol gen.	GQNN	Static topology
[18]	—	Path pred.	GNN+FC	Abstract graphs; greedy
[7]	—	Framework	GN blocks	Foundational
[20]	—	QoS routing	GNN+DRL	RL-based; online interaction
[19]	SDN	Path classif.	Graph-CNN+FC	Centralized; static topology
Ours	LEO	Edge classif.	Enc-Core-Dec	3D, imitation & transfer learning

the necessity of adopting a 3D ECEF representation. Section IV-B makes explicit the shortest-path routing objective that defines the expert solution used for supervision. Section IV-C introduces our primary methodological contribution: formulating the routing problem within an imitation learning paradigm, where the GNN learns to emulate an optimal expert policy via supervised edge classification. Finally, Section IV-D details the GNN architecture adopted in this work.

A. Graph Representation

The satellite network at a given time instant is modeled as a directed graph $G = (V, X_V, E, X_E)$ where V is the set of nodes (satellites and ground stations), X_V is the matrix of node feature vectors, E is the set of directed edges (ISLs and ground-station links), and X_E is the matrix of edge feature vectors.

Each node feature vector $\mathbf{x}_v \in X_V$ encodes the node’s spatial position, together with two binary flags indicating whether the node is the routing *source* or *destination* ground station. Each edge feature $\mathbf{x}_e \in X_E$ contains the straight-line Euclidean distance in kilometers between the connected nodes, which serves as the link cost for shortest-path computation.

Fig. 3 shows a representative Iridium NEXT graph encoded in the traditional 2D geodetic-coordinate representation, where the positional components of $\mathbf{x}_v$ represent the (φ_v, λ_v) coordinates (latitude and longitude, respectively) of node v . As we can observe, geodetic coordinates exhibit a wrap-around discontinuity: longitudes of $\pm 180^\circ$ are numerically distant but spatially adjacent, which can mislead the model into treating nearby satellites as far apart.

To address this issue, we convert satellite positions (φ_v, λ_v) to 3D ECEF coordinates (x_v, y_v, z_v) using the WGS84 ellipsoidal model [21], scaled to $[0, 1]$, thereby increasing the node feature dimension from 4 to 5 components.

B. Shortest-Path Routing

For each topology snapshot and source–destination pair (s, d) , the routing problem is to select a directed path from s to d with minimum total link cost. Let $\mathcal{P}_{s,d}(G)$ denote the set of feasible directed paths from s to d in graph G , and let c_e be the distance-based cost associated with edge e . The expert path is defined as:

$$P_{s,d}^* = \arg \min_{P \in \mathcal{P}_{s,d}(G)} \sum_{e \in P} c_e. \quad (2)$$

The desired output is the edge set $P_{s,d}^* \subseteq E$ that realizes the minimum-cost route for the current topology and routing intent. This path-level objective is later converted into edge-level labels for supervised learning.

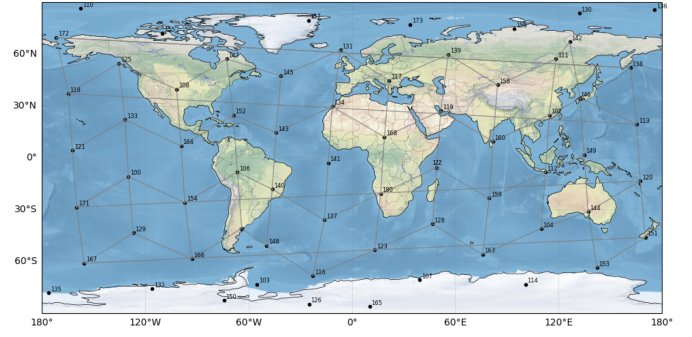


Fig. 3: A representative Iridium NEXT graph in 2D geodetic coordinates. The 66 satellites and 4 ground stations are connected by ISLs and ground-station links. The wrap-around discontinuity at $\pm 180^\circ$ longitude is visible as a gap in the right-hand portion of the layout.

C. Imitation Learning for Supervised Edge Classification

We frame shortest-path routing as an imitation learning problem, where a parametric policy is trained to replicate the decisions of an optimal expert. In our case, the expert is Dijkstra’s algorithm [22], which provides optimal routing decisions in a fully observable setting by solving Eq. (2). Rather than explicitly executing this algorithm at inference time, we train a GNN to amortize its behavior into a feed-forward model that generalizes across unseen topologies.

Concretely, each training sample consists of a graph and the corresponding expert-generated shortest path, which is encoded as binary edge labels $y_e \in \{0, 1\}$ ($y_e = 1$ if edge e belongs to the shortest path, 0 otherwise). This induces a supervised learning problem: given G , the GNN f_ψ produces two edge-level logits $\ell_e = f_\psi(G)_e \in \mathbb{R}^2$, which are converted through a softmax into a path-membership probability $p_e = \Pr(y_e = 1 \mid G)$. A routing decision is then derived by assigning $\hat{y}_e = 1$ when p_e exceeds the decision threshold (0.5): edges predicted as path members form the forwarding route. Conceptually, however, the model learns a policy over graph structures, mapping a global routing objective to distributed edge-level decisions. This perspective is critical: the model is not merely classifying edges, but learning to approximate a combinatorial optimization procedure through data-driven generalization.

Unlike classical imitation learning settings with sequential decision-making [13], our formulation operates in a one-shot, parallel inference regime, where all path-membership probabilities p_e and binary edge decisions $\hat{y}_e$ are produced simultaneously, eliminating error compounding and improving scalability. Moreover, the learned policy implicitly captures structural priors (e.g., spatial locality) that enable robust generalization beyond the expert demonstrations, particularly when combined with synthetic pre-training.

D. GNN Model Architecture

The model follows an *Encoder-Core-Decoder* design, illustrated in Fig. 4 and implemented via the GN framework [7].

1) *Encoder*: The Encoder transforms raw graph features into a shared latent space. It consists of three *independent* Multilayer Perceptrons (MLPs): one for edge features, one for

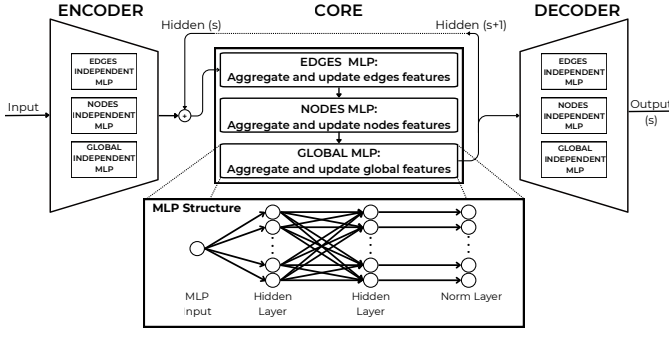


Fig. 4: Encoder-Core-Decoder architecture. The output head (a single linear layer applied to edge representations) is omitted for clarity.

node features, and one for the global graph attribute, operating in parallel without information exchange. Each MLP contains fully connected hidden layers followed by a layer normalization step. Concretely, for each feature type $f \in \{\text{edge, node, global}\}$, the Encoder MLP is:

$$z_f = \text{LayerNorm}\left(W_f^{(L)} \sigma\left(\dots \sigma\left(W_f^{(1)} \mathbf{x}_f + \mathbf{b}_f^{(1)}\right) \dots\right) + \mathbf{b}_f^{(L)}\right), \quad (3)$$

where $\sigma(\cdot) = \text{ReLU}(\cdot)$, $L = 2$ hidden layers are used, and the output dimension is the latent size d . Layer normalization is applied at the output of each MLP (not between hidden layers) to stabilize training without introducing batch-size sensitivity [16]. The three MLPs project inputs of different dimensionality into the same latent space $\mathbb{R}^d$, enabling the Core to exchange information across different representations on a common scale.

2) *Core (Message Passing)*: The Core performs K iterations of message passing on the latent graph produced by the Encoder, following Algorithm 1. At each iteration k , the input to the Core is formed by concatenating the Encoder output $z^{(0)}$ with the latent state from the previous step $z^{(\text{prev})}$ (line 3). This design preserves the original encoder representation throughout all iterations, stabilizing updates and mitigating over-smoothing [16]. We note that, in iteration $k = 0$, where no previous latent state is available, the Encoder output is concatenated with itself to preserve a consistent input dimensionality across all iterations.

Then, three sequential blocks implementing an *Edges MLP* (represented by the update function ϕ^e), *Nodes MLP* (ϕ^v) and *Global MLP* (ϕ^g), respectively, update the graph by aggregating sender/receiver node features to update edge representations (lines 4–6), aggregating incoming edge messages to update node representations (lines 7–10), and pooling node and edge summaries to update the global attribute (line 11). This sequential ordering ensures that information updated in earlier blocks is available to later ones within the same step.

3) *Decoder and Output Head*: Mirroring the Encoder, three independent MLPs refine the latent representations produced by the Core. A final *output head*, i.e., a single fully connected linear layer applied independently to each edge representation, projects the latent space onto two logits ℓ_e for binary classification. The separation of the output head from the Decoder keeps the relational reasoning architecture task-agnostic.

4) *Loss Function*: Since predictions are produced at the end of every message-passing step, the training loss is the cross-

Algorithm 1 Core Message Passing (K iterations)

Require: Encoder output $z^{(0)} = (H_E^{(0)}, H_V^{(0)}, \mathbf{g}^{(0)})$

Require: Number of iterations K , latent dimension d

```

1:  $z^{(\text{prev})} \leftarrow z^{(0)}$  {initialize previous state}
2: for  $k = 0$  to  $K - 1$  do
3:   Core input  $k \leftarrow z^{(0)} \parallel z^{(\text{prev})}$ 
4:   for each edge  $e = (u, v) \in E$  do
5:      $\mathbf{h}_e^{(k)} \leftarrow \phi^e(\mathbf{h}_e^{\text{in}}, \mathbf{h}_u^{\text{in}}, \mathbf{h}_v^{\text{in}}, \mathbf{g}^{\text{in}})$  {edge MLP}
6:   end for
7:   for each node  $v \in V$  do
8:      $\bar{\mathbf{h}}_v \leftarrow \frac{1}{|N(v)|} \sum_{e \in N(v)} \mathbf{h}_e^{(k)}$ 
9:      $\mathbf{h}_v^{(k)} \leftarrow \phi^v(\mathbf{h}_v^{\text{in}}, \bar{\mathbf{h}}_v, \mathbf{g}^{\text{in}})$  {node MLP}
10:  end for
11:   $\mathbf{g}^{(k)} \leftarrow \phi^g(\mathbf{g}^{\text{in}}, \text{mean}(\{\mathbf{h}_e^{(k)}\}), \text{mean}(\{\mathbf{h}_v^{(k)}\}))$  {global MLP}
12:   $z^{(\text{prev})} \leftarrow (H_E^{(k)}, H_V^{(k)}, \mathbf{g}^{(k)})$ 
13:  Compute intermediate loss  $\mathcal{L}^{(k)}$  using output head (Eq. (4))
14: end for
15:  $\mathcal{L} \leftarrow \frac{1}{K} \sum_{k=0}^{K-1} \mathcal{L}^{(k)}$ 
16: return  $z^{(\text{prev})}, \mathcal{L}$ 

```

entropy (CE) between the predicted edge logits and true edge labels, averaged over all K steps:

$$\mathcal{L} = \frac{1}{K} \sum_{k=0}^{K-1} \mathcal{L}^{(k)} = \frac{1}{K} \sum_{k=0}^{K-1} \sum_{e \in E} \text{CE}(\ell_e^{(k)}, y_e). \quad (4)$$

This encourages the model to build progressively better intermediate representations, not only a good final-step prediction.

V. DATA AUGMENTATION AND TRANSFER LEARNING

This section describes our data augmentation and transfer learning strategy, which formally consists of: (1) automatic generation of synthetic graphs with diverse connectivity patterns, both in the 2D and 3D versions described in Section IV-A; (2) pre-training on the synthetic data, and (3) supervised fine-tuning on the real Iridium NEXT topologies.

As mentioned in Section II-A, these exhibit a regular and predictable structure, which may not be common to other LEO constellations. Therefore, synthetic data generation is crucial to prevent the model from overfitting to a specific network architecture, promoting instead robust generalization across different LEO constellations. Accordingly, synthetic graphs are generated by placing $N \in [52, 80]$ nodes in the reference space (matching the Iridium constellation size of 66 satellites plus 4 ground stations) and connecting them via a Geographical Threshold Graph (GTG) [23] with connectivity parameter θ sampled randomly in $[100, 1000]$. In particular, several different θ values are considered to generate each training batch, yielding a diversified dataset that covers scenarios similar, but not identical, to the real network, thereby improving the model's ability to adapt to slightly different configurations. For instance, as exemplified in Fig. 5 for two representative synthetic 2D networks, low- θ graphs are sparse with few long-range edges, while high- θ graphs are dense and resemble planar tessellations. Exposing the model to both extremes encourages robustness to varying edge density at inference time.

Note that the generated GTG for an arbitrary value of θ may not be fully connected [23], which would make it unsuitable for

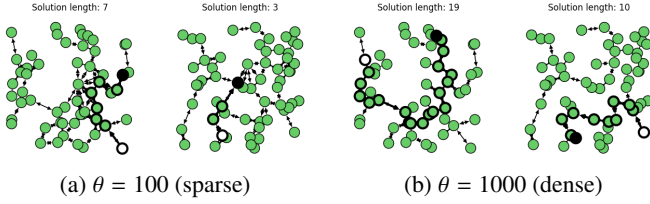


Fig. 5: Synthetic Geographical Threshold Graphs used for pre-training, shown for two values of the connectivity parameter θ . The highlighted edges form the shortest path between the source (white) and destination (black) nodes. Varying θ across batches exposes the model to a wide range of graph densities.

routing tasks. To address this issue, our procedure constructs an auxiliary graph containing all pairs of nodes, weighted by their Euclidean distance, and computes its Minimum Spanning Tree (MST). MST is connected by construction, and can thus be used to determine the minimum number of edges required to connect all nodes. The node attributes of the geographic graph are then transferred to the MST, preserving the information associated with each node. The final graph is obtained by composing the GTG with the MST, so that the edges determined by the parameter θ are preserved while also adding the edges strictly necessary to guarantee global connectivity [24].

Nodes' positions are sampled differently for 2D and 3D models. In the 2D case, nodes are placed at random (x, y) Cartesian coordinates in the unit square. However, real satellites are identified by 2D geodetic coordinates, which parameterize locations on a curved surface. Although distances are computed using a formulation consistent with spherical geometry, pre-training on planar graphs may still bias the model toward Euclidean assumptions that do not fully capture the global structure of spherical graphs. Furthermore, sampling coordinates in $[0, 1]^2$ bounds Euclidean distances to $[0, \sqrt{2}]$, resulting in a narrower distance distribution compared to real graphs. This introduces an extra domain shift between pre-training and fine-tuning, which can negatively affect transfer performance.

In 3D, the nodes of the synthetic graphs are sampled in terms of latitude and longitude over the Earth's surface. Since directly sampling latitude uniformly in the interval $[-90^\circ, 90^\circ]$ would incorrectly concentrate nodes near the poles due to the progressively decreasing circumference of latitude circles, we sample $\sin(\varphi)$ uniformly. This can be interpreted geometrically as the normalized vertical height, ensuring that points are generated within spherical bands of approximately equal surface area, avoiding spatial biases and guaranteeing equal coverage of each geographic region in the dataset. Longitudes, instead, are sampled uniformly in $[-180^\circ, 180^\circ]$. Once generated, the geodetic coordinates (φ, λ) are projected into a three-dimensional Cartesian space by converting them to 3D ECEF coordinates, assuming a constant altitude of 780 km, consistent with the typical orbit of Iridium satellites.

Finally, as for the real datasets, Dijkstra's algorithm is applied to determine the shortest path for all source-destination pairs. Source and destination nodes, in particular, are randomly extracted with the constraint that the resulting path length does not exceed predefined minimum and maximum values. This ensures consistency of the resulting paths with the average path

length expected in a satellite network, preserving the validity of the chosen number of message passing iterations.

VI. EXPERIMENTAL SETUP AND METHODOLOGY

This section presents the experimental framework, detailing the Iridium dataset (Section VI-A), methodology (Section VI-B), hyperparameter selection (Section VI-C), and evaluation metrics used to assess the proposed imitation learning-based routing model (Section VI-D).

A. Iridium NEXT Dataset

Real satellite topologies to be used for model training and validation were sampled from an Iridium NEXT simulator [25] initialized from Two-Line Element (TLE) orbital data (CellesTrak [26], snapshot of 26 October 2023). Topologies were sampled every 10 seconds for one simulated day; four ground stations (Tokyo, New York, Dar es Salaam, San Francisco) yielded 12 source-destination (SD) pairs per topology snapshot. The resulting dataset contains 103,680 labeled graph pairs, 30% of which were retained for validation.

A separate test set of another 103,680 graphs is generated from a different TLE file (6 October 2025) to ensure that no topology seen during training appears at test time. The choice of two TLE snapshots separated by two years is deliberate: because Iridium NEXT orbits are nearly-repeating (the constellation has an exact repeat cycle of approximately 24 hours), consecutive-day snapshots would contain nearly identical satellite configurations, which would not constitute a genuine train/test split. By using snapshots from different calendar years, we ensure that the ground-station pass geometry, ISL activation patterns, and topology distributions in the test set are genuinely unseen.

The ground stations were selected to maximize geographic diversity and to cover a representative range of SD arc lengths: Tokyo to San Francisco spans the Pacific (approximately 8280 km), while New York to Dar es Salaam spans the Atlantic and equatorial Africa (approximately 11,400 km). Paths across both oceanic and continental links are thus represented in the dataset, covering the main operational scenarios for transoceanic satellite communication. The 12 unique SD pairs per snapshot (including both directions for each pair) ensure that each topology contributes samples with different routing objectives.

Positional node features were scaled to $[0, 1]$ via *min-max* normalization; edge distances were standardized using *z-score* normalization over global statistics.

B. Experimental Methodology

We conducted an ablation study to evaluate our data augmentation and transfer learning pipeline, and the impact of adopting a 3D representation, developing four model variants in a progressive way as summarized in Table II. All models share the *Encoder-Core-Decoder* architecture presented in Section IV-D, while they differ in input representation and training strategy.

In particular, *Satellite 2D* and *3D* models were trained directly on 72,576 real Iridium NEXT graphs considering a batch size of 100 (other hyperparameters were tuned and selected as discussed in the next section). On the other hand, the pre-training

TABLE II: Summary of model variants.

Model	Representation	Training
Satellite 2D	2D geodetic (φ, λ)	Real data only
Fine-tuned 2D	(x, y)	Synthetic $\rightarrow$ Fine-tune
Satellite 3D	3D ECEF (x, y, z)	Real data only
Fine-tuned 3D	3D ECEF (x, y, z)	Synthetic $\rightarrow$ Fine-tune

TABLE III: Optimal hyperparameters for 2D and 3D models.

Hyperparameter	2D models	3D models
Latent MLP dimension (d)	32	64
MLP hidden layers	2	2
Message-passing steps (K)	12	15
Learning rate (training)	10^{-3}	10^{-3}
Learning rate (fine-tuning)	10^{-4}	10^{-4}

phase for the *Fine-tuned 2D* and *3D* models was conducted on 500,000 synthetic graphs generated as described in Section V (considering a validation set of 50,000 samples), and followed by a *fine-tuning* on the same 72,576 satellite graphs as the baseline.

C. Hyperparameter Selection

Hyperparameters for each model variant were selected via the Hyperopt library [27] using Tree-structured Parzen Estimators. The search space included latent MLP dimension (d), number of hidden layers per MLP, number of message-passing steps (K), and learning rate. Table III reports the optimized values for the two distinct configurations identified by this search.

Increasing d from 32 to 64 and K from 12 to 15 is motivated by the higher input dimensionality (5 vs. 4 node features) and the greater geometric complexity of the 3D spatial domain. Larger latent spaces and more message-passing steps allow the model to build richer representations of the satellite mesh without over-smoothing, while the fine-tuning learning rate of 10^{-4} was chosen to be one order of magnitude below the pre-training rate, following standard transfer learning practice [28].

All experiments are repeated with 5 random seeds; results are reported as mean $\pm$ standard error of the mean (SEM).

D. Evaluation Methodology and Accuracy Metrics

Each model is evaluated on the corresponding test set; for fine-tuned models, checkpoints are saved at 0, 20, 40, 60, 100, 300, 500 fine-tuning iterations and at the end of full fine-tuning to analyze data efficiency.

We considered three groups of metrics. Let $\mathcal{G} = \{G_i\}_{i=1}^I$ be the set of validation or test graphs, and let $\hat{P}_i$ and P_i^* denote the predicted and ground-truth edge sets for graph G_i .

The first group of metrics, denoted as **classification metrics**, were collected during both the validation and the test phase. They are: (1) the *average loss*, i.e., the mean cross-entropy over all edge predictions, averaged over the I graphs, which provides a general measure of the discrepancy between the model predictions and the true values; (2) the *fraction correct* (FC), defined as the fraction of edges whose predicted label matches the ground-truth label, averaged over graphs:

$$\text{FC} = \frac{1}{N} \sum_{i=1}^N \frac{|\{e \in E_i : \hat{y}_e = y_e\}|}{|E_i|}. \quad (5)$$

By quantifying the proportion of edges correctly classified with respect to the total, this metric indicates how well the

model captures local relationships in the graph. Finally, (3) the *fraction solved* (FS) measures the fraction of graphs for which the model predicts the exact shortest path, i.e., $\hat{P}_i = P_i^*$:

$$\text{FS} = \frac{1}{N} \sum_{i=1}^N \mathbb{1}[\hat{P}_i = P_i^*]. \quad (6)$$

By counting the number of graphs for which the model provides a completely correct solution, FS represents the model's ability to solve routing problems in their entirety.

Moreover, during the test phase we monitored **per-class metrics** including average precision, recall, and F1-score:

$$\text{Precision} = \frac{|\hat{P}_i \cap P_i^*|}{|\hat{P}_i|}, \quad \text{Recall} = \frac{|\hat{P}_i \cap P_i^*|}{|P_i^*|}, \quad (7)$$

while F1 is their harmonic mean, computed over the binary path-membership labels, and macro-averaged over both classes. These evaluate the prediction quality on unseen data, highlighting imbalances and providing insight not only into how accurately the model distinguishes edges belonging or not to the shortest path, but also into its reliability in capturing all edges actually involved, confirming the generalizability of the performance observed during training.

Finally, we monitored **path-level metrics** that are computed on *incorrect* predictions $\hat{P}_i \neq P_i^*$ (unless stated as *global*). These are: (1) the *Jaccard similarity* $J_i = |\hat{P}_i \cap P_i^*| / |\hat{P}_i \cup P_i^*|$, which measures overlap between predicted and target paths; in particular, $J_i = 1$ only when $\hat{P}_i = P_i^*$. (2) The *average stretch*, defined as the ratio of predicted to optimal path cost $c(\hat{P}_i)/c(P_i^*)$, where $c(\cdot)$ sums Euclidean edge distances; values below 1 indicate truncated paths, values above 1 indicate longer routes. (3) The *average path length difference*, defined as the mean difference in hop count between $\hat{P}_i$ and P_i^* . Finally, (4) the *feasible rate* (FR), i.e., the fraction of predicted paths that form a valid source-to-destination route (i.e., a connected path); this is the only path-level metric we report both on wrong predictions (*FR wrong*) and on the full test set (*FR global*). Indeed, the former allows evaluating, in cases where the model does not identify the correct shortest path, how often the predicted solutions are still feasible. The latter, instead, provides an overall measure of the model's ability to generate feasible paths across all examples considered. In a potential routing application, this distinction is crucial, as the main objective is not solely to identify the shortest path, but also to ensure that packets actually reach the destination, even if along suboptimal routes.

VII. EXPERIMENTAL RESULTS

This section reports the results of our experiments, observing the impact of the transfer learning procedure we propose and the adoption of a 3D representation model as discussed in Section VI-B (Sections VII-A–VII-C), and proposing a qualitative comparison against classical baselines (Section VII-D).

A. Pre-training Convergence

This section evaluates the convergence dynamics during the pre-training phase conducted, for both 2D and 3D models, on synthetic data generated as discussed in Section V.

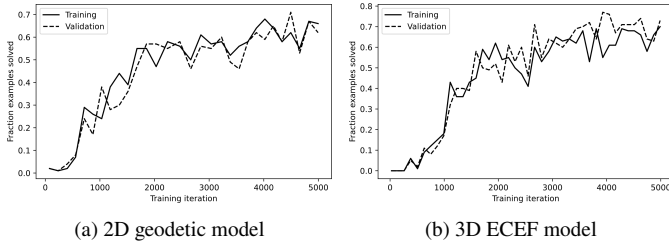


Fig. 6: FS during synthetic pre-training. The model learns general shortest-path structure on synthetic spatial graphs before any satellite-specific fine-tuning.

The value of FS for the two cases is reported in Fig. 6. We can observe that it rises from near zero to approximately 55% within the first 2000 iterations and stabilises around 70% by the end of pre-training. This demonstrates that the models successfully learn a general notion of shortest-path structure in spatial graphs even without observing satellite-specific data. The 2D model performs slightly worse than those of the 3D model, which aligns with the observation that the 3D ECEF representation is intrinsically more regular (no wrap-around discontinuity even in the unit-square encoding for the y -coordinate) and provides a richer geometric signal.

B. Impact of Transfer Learning and 3D Representation

A comprehensive comparison of the FS, F1-score and FR metrics as a function of the fine-tuning iterations for the four model variants listed in Table II is reported in Fig. 7, while Tables IV and V report the full metric set at the most informative checkpoints (mean $\pm$ SEM). To improve clarity, the optimization direction for each metric is reported in parenthesis ($\uparrow$ if higher is better); moreover, for each metric, we highlight in green the checkpoints at which the fine-tuned model outperforms the baseline on average². We can observe that this occurs after at most 100 fine-tuning iterations for the 2D model and around 20 iterations for the 3D model, demonstrating the effectiveness of our transfer learning approach.

To discuss the impact of our training strategies, we first analyze the convergence dynamics of the two baseline models (Satellite 2D and Satellite 3D in Table II), observing the effects of a 3D representation, and then evaluate transfer learning.

1) *Impact of 3D representation:* The convergence dynamics of Satellite models (in the 2D and 3D variants) in terms of the FS and FC metrics are reported in Fig. 8.

The Satellite 3D model exhibits faster convergence on both metrics, and FS also reaches a higher plateau, saturating near 82.8% while the Satellite 2D model never exceeds 70%. Both models show a characteristic “phase transition” behavior: while the per-edge accuracy (FC) rises smoothly from the start, FS, which requires *all* edges to be correctly classified, remains near zero until FC crosses a critical threshold (around iteration 100 for the 2D model, which corresponds to nearly 10,000 processed graphs), after which it rises sharply. This non-linear relationship

²We note that it is not possible to define an explicit optimization direction for the average stretch metric, which is optimal if equal to 1 and sub-optimal both above and below. However, we consider values slightly above 1 to be “better” than values below 1, since the former denotes the presence of longer but feasible paths in the solution, while the latter denotes infeasible, truncated paths.

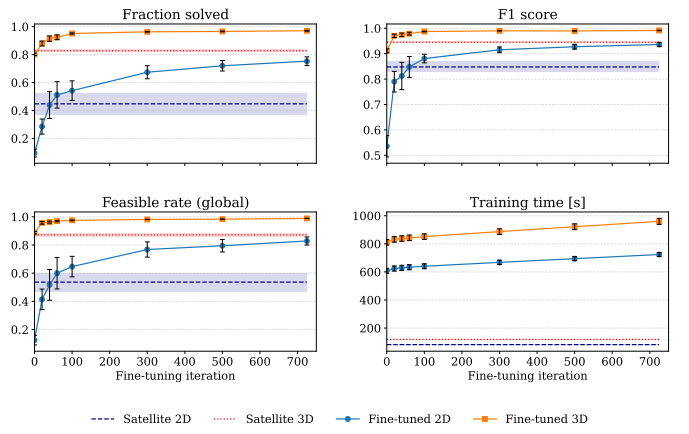


Fig. 7: Models comparison across fine-tuning iterations. Horizontal lines show models trained on satellite data without fine-tuning. Standard errors are shown as shaded bands or error bars. Fine-tuned training time includes pre-training.

between edge-level and path-level accuracy is intrinsic to the edge classification formulation and motivates reporting both metrics. Quantitatively, for the Satellite 3D model, FC rises above 98% before FS exceeds 50%. This is because a typical shortest path contains 5 to 7 edges, so even a single wrong prediction (per-edge error rate $< 2\%$) causes the entire path to be marked incorrect. As training proceeds, per-edge accuracy rises above 99%, at which point the probability that all ℓ path edges are correct, approximated as $(0.99)^\ell$, is approximately 93%–95%, consistent with the observed FS plateau of 82.8% (the gap is explained by variance across seeds and the distribution of longer paths). This analysis underlines that FS is a *strictly harder* and more operationally meaningful metric than FC: a router with 99% per-edge accuracy still misdirects roughly one in seven packets on a 6-hop path.

Considering the results on the test set reported in Tables IV and V, we observe that, for the 2D model, the precision (92.3%) is substantially higher than recall (81.4%), indicating that the model tends to predict *fewer* path edges than the true set. This systematic under-prediction bias is consistent with the wrap-around artifact: satellites near the $\pm 180^\circ$ seam are assigned anomalously large geodetic distances, causing the model to avoid selecting ISLs in that region. This produces truncated paths (the average stretch is 0.78, meaning that predicted paths are 22% shorter in total cost than the optimal path, corresponding to missing links rather than overly long paths) and low recall. In the 3D case, precision (95.8%) and recall (93.9%) are nearly balanced, and an average stretch of 0.874 still indicates mild truncation, but at a much lower severity than the 2D baseline. This suggests that the 3D representation resolves the primary source of systematic error, but that model capacity or training data quantity remain limiting factors.

2) *Impact of transfer learning:* The convergence dynamics for the 2D and 3D models during fine-tuning in terms of the FS and FC metrics are reported in Fig. 9. We can observe that, as already mentioned when comparing the results on the test set reported in Tables IV and V, pre-training on synthetic data consistently benefits both representations.

TABLE IV: Evaluation metrics for 2D models on the test set (mean $\pm$ SEM over 5 seeds). Fine-tuned 2D training times include pre-training.

Metric	Satellite 2D	Fine-tuned 2D (iterations)				
		0	20	100	300	Full
Avg. Loss ($\downarrow$)	0.030 $\pm$ 0.003	0.177 $\pm$ 0.025	0.042 $\pm$ 0.011	0.024 $\pm$ 0.002	0.018 $\pm$ 0.002	0.014 $\pm$ 0.002
Fraction Correct ($\uparrow$)	0.989 $\pm$ 0.001	0.952 $\pm$ 0.010	0.983 $\pm$ 0.004	0.991 $\pm$ 0.001	0.993 $\pm$ 0.001	0.995 $\pm$ 0.000
Fraction Solved ($\uparrow$)	0.448 $\pm$ 0.076	0.095 $\pm$ 0.028	0.285 $\pm$ 0.054	0.542 $\pm$ 0.071	0.674 $\pm$ 0.047	0.753 $\pm$ 0.031
Avg. Precision ($\uparrow$)	0.923 $\pm$ 0.015	0.564 $\pm$ 0.066	0.859 $\pm$ 0.048	0.925 $\pm$ 0.005	0.931 $\pm$ 0.006	0.947 $\pm$ 0.004
Avg. Recall ($\uparrow$)	0.814 $\pm$ 0.022	0.586 $\pm$ 0.039	0.768 $\pm$ 0.034	0.861 $\pm$ 0.025	0.910 $\pm$ 0.017	0.933 $\pm$ 0.011
Avg. F1-score ($\uparrow$)	0.848 $\pm$ 0.020	0.536 $\pm$ 0.042	0.790 $\pm$ 0.041	0.881 $\pm$ 0.017	0.915 $\pm$ 0.012	0.936 $\pm$ 0.008
Jaccard (wrong) ($\uparrow$)	0.598 $\pm$ 0.014	0.343 $\pm$ 0.028	0.586 $\pm$ 0.038	0.619 $\pm$ 0.004	0.623 $\pm$ 0.010	0.625 $\pm$ 0.008
Avg. Stretch (wrong)	0.776 $\pm$ 0.034	1.374 $\pm$ 0.290	0.876 $\pm$ 0.053	0.878 $\pm$ 0.047	0.965 $\pm$ 0.060	0.960 $\pm$ 0.028
Feasible Rate (global) ($\uparrow$)	0.536 $\pm$ 0.067	0.125 $\pm$ 0.034	0.414 $\pm$ 0.073	0.647 $\pm$ 0.073	0.768 $\pm$ 0.054	0.828 $\pm$ 0.029
Avg. Path Len. Diff. ($\downarrow$)	2.455 $\pm$ 0.138	4.568 $\pm$ 1.288	2.365 $\pm$ 0.167	1.932 $\pm$ 0.166	1.689 $\pm$ 0.056	1.557 $\pm$ 0.112
Training Time [s]	81.9 $\pm$ 1.7	607.2 $\pm$ 18.3	624.2 $\pm$ 18.4	640.2 $\pm$ 17.9	668.0 $\pm$ 15.8	723.9 $\pm$ 13.9

TABLE V: Evaluation metrics for 3D models on the test set (mean $\pm$ SEM over 5 seeds). Fine-tuned 3D training times include pre-training.

Metric	Satellite 3D	Fine-tuned 3D (iterations)			
		0	20	100	Full
Avg. Loss ($\downarrow$)	0.012 $\pm$ 0.000	0.031 $\pm$ 0.004	0.007 $\pm$ 0.001	0.004 $\pm$ 0.000	0.003 $\pm$ 0.000
Fraction Correct ($\uparrow$)	0.996 $\pm$ 0.000	0.992 $\pm$ 0.001	0.997 $\pm$ 0.000	0.999 $\pm$ 0.000	0.999 $\pm$ 0.000
Fraction Solved ($\uparrow$)	0.828 $\pm$ 0.008	0.802 $\pm$ 0.008	0.880 $\pm$ 0.017	0.951 $\pm$ 0.004	0.971 $\pm$ 0.003
Avg. Precision ($\uparrow$)	0.958 $\pm$ 0.002	0.912 $\pm$ 0.009	0.969 $\pm$ 0.006	0.987 $\pm$ 0.001	0.991 $\pm$ 0.001
Avg. Recall ($\uparrow$)	0.939 $\pm$ 0.006	0.915 $\pm$ 0.007	0.974 $\pm$ 0.007	0.988 $\pm$ 0.001	0.992 $\pm$ 0.000
Avg. F1-score ($\uparrow$)	0.945 $\pm$ 0.003	0.912 $\pm$ 0.008	0.970 $\pm$ 0.007	0.987 $\pm$ 0.001	0.991 $\pm$ 0.001
Jaccard (wrong) ($\uparrow$)	0.556 $\pm$ 0.008	0.431 $\pm$ 0.027	0.650 $\pm$ 0.017	0.623 $\pm$ 0.011	0.584 $\pm$ 0.011
Avg. Stretch (wrong)	0.874 $\pm$ 0.050	1.075 $\pm$ 0.039	1.074 $\pm$ 0.026	1.017 $\pm$ 0.013	1.044 $\pm$ 0.010
Feasible Rate (global) ($\uparrow$)	0.872 $\pm$ 0.013	0.887 $\pm$ 0.008	0.958 $\pm$ 0.010	0.975 $\pm$ 0.003	0.990 $\pm$ 0.001
Avg. Path Len. Diff. ($\downarrow$)	1.892 $\pm$ 0.205	1.597 $\pm$ 0.085	1.430 $\pm$ 0.071	1.211 $\pm$ 0.031	1.074 $\pm$ 0.032
Training Time [s]	119.2 $\pm$ 1.7	810.5 $\pm$ 19.8	831.4 $\pm$ 19.7	851.6 $\pm$ 20.0	960.2 $\pm$ 21.6

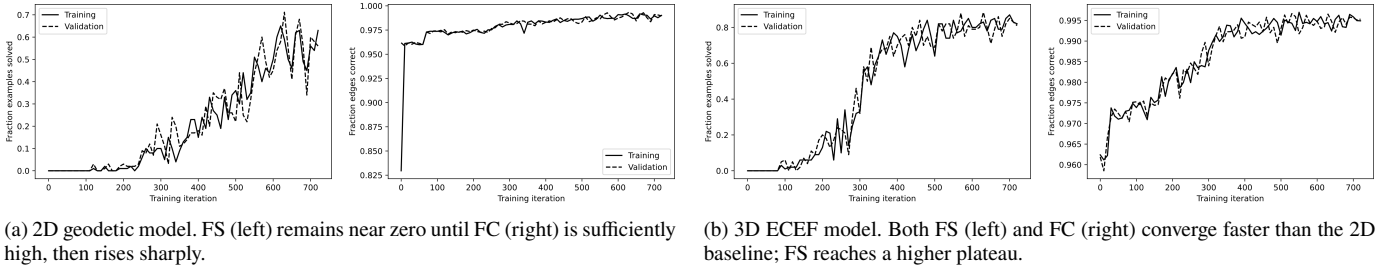


Fig. 8: Fraction Solved and Fraction Correct during training of the Satellite models.

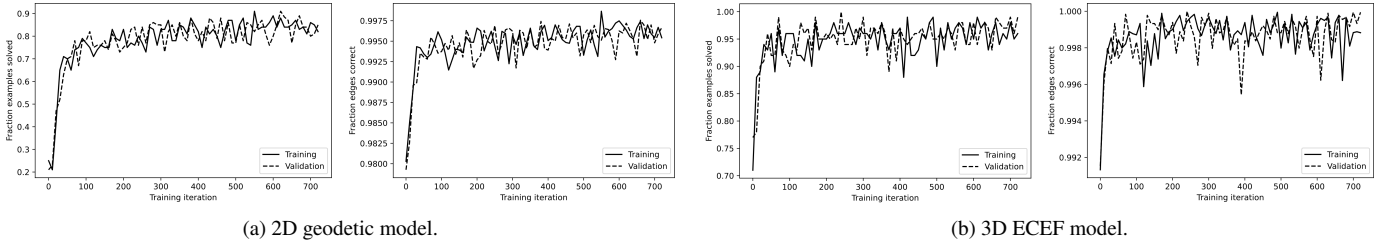


Fig. 9: Fraction Solved and Fraction Correct during fine-tuning.

For the 3D model, the model yields above 70% FS even before any satellite-specific fine-tuning (iteration 0), whereas the Satellite 3D model trained directly on real data achieves 82.8% only after full training (Fig. 8b). Fine-tuning then rapidly pushes the pre-trained model to a FS of 97.1% within 500 iterations. In the 2D case, the benefit is even more pronounced: the baseline Satellite 2D reaches only around 60% (Fig. 8a), while fine-tuning the pre-trained model yields slightly above 80%.

Even when a model fails to reconstruct the exact shortest path, the predicted path is typically near-optimal: for the fine-tuned 3D model evaluated on the test set (Table V), the Jaccard similarity score on wrong predictions exceeds 0.58, the average stretch is close to 1.04 (predicted paths are slightly longer than optimal),

and the average path length difference is approximately 1.1 hops, confirming that errors are confined to routes that deviate by at most one or two hops. For the 2D models (Table IV), errors are more severe: the stretch is below 1, still denoting the presence of truncated paths, and the path length differences is around 1.6 hops after fine-tuning, consistent with the lower recall values (93.3% for fine-tuned 2D vs. 99.2% for fine-tuned 3D).

The global FR mirrors the FS trend. The fine-tuned 3D model achieves 99.0%, meaning that less than 1% of predictions result in an undeliverable packet, providing a strong guarantee for a learning-based router. Overall, these numbers support the conclusion that the 3D model effectively learns the routing structure of the Iridium NEXT constellation.

C. Training time and data efficiency

All training experiments discussed in the previous section were conducted on a VM equipped with 8 Intel(R) Xeon(R) Silver vCPUs, 1 NVIDIA TITAN Xp GPU and 32Gb of RAM.

Both pre-training phases take under 15 minutes on a single GPU, making the cost of generating and training on 500,000 synthetic graphs negligible relative to the performance gain induced by transfer learning. Notably, the most substantial performance gains occur within the first 100 fine-tuning iterations (approximately 150 seconds of GPU time for the 3D model). For the 3D model, FS increases from 80.2% to 95.1%, while the remaining iterations only add 2.0 percentage points (pp), leading FS to 97.1%, and 1.5 pp on global FR (to 99.0%), at a negligible additional cost of approximately 109 seconds.

D. Comparison with Classical Baselines

For completeness, we note that Dijkstra’s algorithm, the expert used to generate training labels, achieves 100% Fraction Solved and Feasible Rate by definition, at the cost of $O(|E| + |V| \log |V|)$ per query. For the 70-node Iridium NEXT graphs ($|V| = 70$, $|E| \approx 280$), this translates to approximately 10^4 floating-point operations per shortest-path computation. The GNN forward pass requires a fixed number of matrix multiplications (determined by K , d , and the number of edges), but operates on the full graph in parallel. While a direct runtime comparison is outside the scope of this work, the key structural distinction is that the GNN forward pass operates on the entire graph in parallel (all edges are scored simultaneously in a single pass), whereas Dijkstra processes nodes sequentially. Furthermore, in a distributed satellite setting, the Encoder and Core need only run once per topology snapshot on the ground; each satellite then applies a lightweight local Decoder to make per-hop decisions without requiring global state. By contrast, Dijkstra requires the full routing table to be distributed to every node after each topology change.

VIII. HIGHLIGHTS AND DISCUSSION

Representation matters. Switching from 2D geodetic to 3D ECEF coordinates eliminates a non-physical numerical artifact at $\pm 180^\circ$. Without this correction, the Encoder learns flawed spatial relationships near the seam. ECEF coordinates provide a geometrically faithful representation from the first training iteration, making this our most impactful design choice.

Transfer learning reduces satellite data requirements. While training directly on Iridium NEXT achieves acceptable performance (82.8% FS), pre-training on 500,000 synthetic graphs unlocks an 80.2% initialization *before* seeing real topology, while fine-tuning adapts the model in under 200 GPU-seconds. This dramatically simplifies generalizing to new constellations by requiring only small specific fine-tuning sets.

Practical routing implications. A 99.0% global FR guarantees that less than 1 in 100 packets fail due to model error. Furthermore, when errors occur, paths remain near-optimal (stretch ≈ 1.04 , length difference ≈ 1.1 hops), with the model consistently identifying marginally longer but feasible routes.

Training efficiency and cost-benefit analysis. The best-performing model trains in ≈ 960 GPU-seconds (810 s synthetic pre-training, 150 s fine-tuning), yielding 97.1% FS compared to 82.8% for direct satellite training (120 s). Crucially, pre-training is a one-time investment; adapting to new constellations requires only the fine-tuning phase. Furthermore, the first 100 fine-tuning iterations (40 s) yield the bulk of the gain (14.9 pp), while the remaining 400 iterations add only 2.0 pp. This diminishing-returns behavior makes early stopping at iteration 100 a highly cost-effective deployment heuristic.

Limitations. The framework currently ignores traffic load, link congestion, and localized link failures not captured in the topology snapshot. Moreover, experiments target the Iridium NEXT constellation; scaling to denser networks as, e.g., Starlink (~ 10000 satellites) requires further investigation of inference latency, memory footprint, and fully decentralized execution.

Towards distributed deployment. The Encoder-Core-Decoder architecture naturally enables splitting the execution between ground and satellite segments to minimize on-board computation. Periodically, a ground station can collect TLE data, execute Encoder and Core to produce latent graph embeddings, and broadcast them to all satellites. In this scheme, satellites do not maintain an explicit representation of the global constellation topology, requiring only the precomputed embeddings together with knowledge of their currently available ISLs. Upon receiving a packet, each satellite queries its local lightweight Decoder, using the precomputed embeddings and packet headers to select the output port at line rate via fixed-size matrix operations (e.g., on FPGAs). This trades periodic downlink overhead for near-zero forwarding latency, eliminating the need for on-board routing tables or distributed consensus. The corresponding on-board inference memory footprint is approximately 0.41MB for the model, 771kB for the latent embeddings (in the considered scenario), plus a negligible overhead for lightweight indexing structures.

IX. CONCLUSION

We presented a GNN-based framework for shortest-path LEO satellite routing, formulated as supervised binary edge classification. Combining an Encoder-Core-Decoder architecture, 3D ECEF representations, and a transfer learning pipeline, our approach achieves a 97.1% FS and 99.0% FR on Iridium NEXT constellation. Crucially, 3D coordinates eliminate wrap-around artifacts, while transfer learning drastically accelerates convergence and ensures occasional errors remain near-optimal.

Future work will incorporate traffic-aware objectives and validate the distributed ground-satellite execution model. Moreover, to mitigate challenges as quadratic edge growth arising when scaling to mega-constellations, we will explore sparse message passing, hierarchical GNNs, and multi-scale pre-training. Importantly, our transfer learning pipeline inherently simplifies this transition, requiring only constellation-specific fine-tuning data without altering the core architecture.

REFERENCES

- [1] J. Lorincz and Z. Klarin, "How trend of increasing data volume affects the energy efficiency of 5g networks," *Sensors*, vol. 22, no. 1, p. 255, Dec. 2021. [Online]. Available: <https://www.mdpi.com/1424-8220/22/1/255>
- [2] O. Kodheli, E. Lagunas, N. Maturo, B. Shankar, S. K. Sharma, J. F. M. Montoya, J. C. M. Duncan, D. Spano, S. Chatzinotas, S. Kisseleff, J. Querol, L. Lei, T. X. Vu, and G. Goussetis, "Satellite communications in the new space era: A survey and future challenges," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 1, First Quarter 2021.
- [3] B. Soret, I. Leyva-Mayorga, M. Röper, D. Wübben, B. Matthiesen, A. Dekorsy, and P. Popovski, "Leo small-satellite constellations for 5g and beyond-5g communications," *arXiv*, 2020, version 3, 22 Jun 2020. [Online]. Available: <https://arxiv.org/abs/1912.08110>
- [4] Fortune Business Insights, "Satellite Internet Market Size, Share & Industry Analysis," <https://www.fortunebusinessinsights.com/satellite-internet-market-109242>, 2026, visited: 1/4/2026.
- [5] B. Soret and D. Smith, "Autonomous Routing for LEO Satellite Constellations with Minimum Use of Inter-Plane Links," in *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, 2019, pp. 1–6.
- [6] W. Jiang, H. Han, Y. Zhang, J. Wang, M. He, W. Gu, J. Mu, and X. Cheng, "Graph neural networks for routing optimization: Challenges and opportunities," *Sustainability*, vol. 16, no. 21, 2024. [Online]. Available: <https://www.mdpi.com/2071-1050/16/21/9239>
- [7] 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*, Oct 17 2018, cs.LG. [Online]. Available: <https://arxiv.org/abs/1806.01261>
- [8] A. Singh, S. Sharma, A. Gumaste, and J. Wang, "Grafnet: Using graph neural networks to create table-less routers," *IEEE Transactions on Network Science and Engineering*, vol. 9, no. 2, pp. 740–755, Mar-Apr 2022. [Online]. Available: <https://doi.org/10.1109/TNSE.2021.3132070>
- [9] F. Geyer and G. Carle, "Learning and generating distributed routing protocols using graph-based deep learning," in *Proceedings of the ACM SIGCOMM 2018 Workshop on Big Data Analytics and Machine Learning for Data Communication Networks (Big-DAMA'18)*. Budapest, Hungary: ACM, Aug 20 2018, pp. 1–6. [Online]. Available: <https://doi.org/10.1145/3229607.3229610>
- [10] M. Liu, J. Li, and H. Lu, "Routing in small satellite networks: A gnn-based learning approach," *arXiv preprint arXiv:2108.08523*, Aug 2021, cs.NI. [Online]. Available: <https://arxiv.org/abs/2108.08523>
- [11] S. S. Jamal and W.-C. Song, "Gnn-based routing for link reliability optimization in sd-leo satellite networks," in *Proceedings of the IEEE Conference on NFV-SDN*, Nov 2024. [Online]. Available: <https://doi.org/10.1109/NFV-SDN61811.2024.10807492>
- [12] Iridium Constellation LLC, "Iridium next engineering statement," Online, Dec. 2013, <https://fcc.report/IBFS/SAT-MOD-20131227-00148/1031348.pdf>.
- [13] B. Zheng, S. Verma, J. Zhou, I. W. Tsang, and F. Chen, "Imitation learning: Progress, taxonomies and challenges," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 35, no. 5, pp. 6322–6337, 2022.
- [14] C. E. Fossa, R. A. Raines, G. H. Gunsch, and M. A. Temple, "An overview of the ifidiumgo low earth orbit (leo) satellite system," Department of Electrical and Computer Engineering, Air Force Institute of Technology, 2950 P St, Bldg. 640, Wright-Patterson AFB, OH 45433-7765, USA, Technical Report, 1998.
- [15] RVMobileInternet.com, "Iridium constellation iii [online image]," https://www.rvmobileinternet.com/wp-content/uploads/2015/09/SPAC_Iridium_Constellation_III_Ig.gif, accessed: 2026-01-12.
- [16] W. L. Hamilton, *Graph Representation Learning*, ser. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2020, vol. 14, no. 3.
- [17] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph Attention Networks," in *International Conference on Learning Representations*, 2018. [Online]. Available: <https://openreview.net/forum?id=rJXMpikCZ>
- [18] B. S. Neyigapula, "Graph neural networks for optimal pathfinding: Uncovering the shortest distances," *Research Square*, Aug 16 2023. [Online]. Available: <https://doi.org/10.21203/rs.3.rs-3249175/v1>
- [19] Z. Zhuang, J. Wang, Q. Qi, H. Sun, and J. Liao, "Toward greater intelligence in route planning: A graph-aware deep learning approach," *IEEE Systems Journal*, vol. 14, no. 2, pp. 1658–1669, Jun 2020. [Online]. Available: <https://doi.org/10.1109/JSYST.2019.2922217>
- [20] H. Wei, Y. Zhao, and K. Xu, "G-routing: Graph neural networks-based flexible online routing," *IEEE Network*, vol. 37, no. 4, pp. 90–95, Jul-Aug 2023. [Online]. Available: <https://doi.org/10.1109/MNET.012.2300052>
- [21] N. Imagery and M. Agency, "Department of defense world geodetic system 1984: Its definition and relationships with local geodetic systems," NIMA, Tech. Rep. TR8350.2, Jan 2000.
- [22] N. Makariye, "Towards shortest path computation using dijkstra algorithm," in *2017 International Conference on IoT and Application (ICIOT)*. Nagapattinam, India: IEEE, May 2017.
- [23] N. Masuda, H. Miwa, and N. Konno, "Geographical threshold graphs with small-world and scale-free properties," *Physical Review E*, vol. 71, p. 036108, 2005.
- [24] M. Bradonjić, A. Hagberg, and A. G. Percus, "Giant component and connectivity in geographical threshold graphs," in *Algorithms and Models for the Web-Graph (WAW 2007)*, A. Bonato and F. Chung, Eds., 2007, pp. 209–216.
- [25] "Satellite Routing Simulator," 2024, accessed: 2026-04-24. [Online]. Available: <https://gitlab.com/MatteoRondena/satellite-routing-simulator>
- [26] T. S. Kelso, "Celestrak: Satellite orbit data and tools," 2026, accessed: 2026-01-28. [Online]. Available: <https://celestrak.org/>
- [27] J. Bergstra, D. Yamins, and D. D. Cox, "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures," in *Proceedings of the 30th International Conference on Machine Learning (ICML)*, June 2013, pp. 115–123.
- [28] H. Li, P. Chaudhari, H. Yang, M. Lam, A. Ravichandran, R. Bhotika, and S. Soatto, "Rethinking the hyperparameters for fine-tuning," in *International Conference on Learning Representations (ICLR)*, 2020, published as a conference paper at ICLR 2020, arXiv:2002.11770v1 [cs.CV].