Ingénierie des Systèmes d'Information
Vol. 31, No. 2, February, 2026, pp. 567-577
Journal homepage: http://iieta.org/journals/isi

A Graph Neural Network Framework for Automated Intent-to-Policy Translation and Enforcement in Software-Defined Networks

Oluwabukola F. Ajayi[1*], Afolashade O. Kuyoro[1], Aaron A. Izang[2], Folasade Y. Ayankoya[1], Oyebola Akande[1], Alfred A. Udosen[1], Oluwole Solanke[1], Bamidele Doyin[1], Afolarin I. Amusa[1], Yaw A. Mensah[3]

[1] Department of Computer Science, Babcock University, Ilishan-Remo 121103, Nigeria
[2] Department of Information Technology, Babcock University, Ilishan-Remo 121103, Nigeria
[3] Department of Software Engineering, Babcock University, Ilishan-Remo 121103, Nigeria

Corresponding Author Email: ajayioluwa@babcock.edu.ng

https://doi.org/10.18280/isi.310224

Received: 26 November 2025
Revised: 25 January 2026
Accepted: 19 February 2026
Available online: 28 February 2026

Keywords:
Graph Neural Networks, Intent-Based Networking, Software-Defined Networking, intent-to-policy translation, policy enforcement, network automation, topology-aware learning, real-time inference

ABSTRACT

As modern networks grow in scale and complexity, translating high-level user intents into low-level enforceable policies remains a fundamental challenge in Software-Defined Networking (SDN). While Intent-Based Networking (IBN) enables declarative specification of network behavior, existing approaches largely rely on static rules or task-specific models, limiting their adaptability and generalization. This study proposes an end-to-end Graph Neural Network (GNN) framework for automated intent-to-policy translation and enforcement in SDN environments. The network is modeled as a dynamic graph and jointly embedded with structured user intents using attention-based GNN architectures, including Graph Attention Networks and GraphSAGE. The learned representations directly generate controller-ready policy vectors that support multiple intent types, such as performance, reachability, isolation, and security. The framework integrates intent parsing, GNN-based inference, and real-time policy enforcement within a unified pipeline. Experimental results on real and synthetic network topologies demonstrate significant improvements in translation accuracy, adaptability to dynamic conditions, and enforcement latency compared to rule-based and conventional machine learning approaches. These findings highlight the effectiveness of GNNs as a unified representation learning paradigm for autonomous intent-driven network control.

1. INTRODUCTION

The rise of Software-Defined Networking (SDN) has redefined how modern networks are designed, controlled, and managed. By decoupling the control plane from the data plane, SDN allows centralized, programmable control over network behavior [1]. Nevertheless, with increasingly complex networks and more flexible user requirements, conventional SDN policy enforcement systems, which are usually based on fixed rule-based settings, cannot scale or dynamically respond to real-time requirements [2].

Intent-Based Networking (IBN) has become a paradigm shift enabling the administrator to articulate what is desired by the network to accomplish such as quality of service (QoS) guarantees, traffic isolation or access controls without describing how it is to be accomplished [3]. This exceptionally abstract representation greatly eases the management of the network but presents new challenges: the interpretation of human readable intent into low-level enforceable rules is a mostly manual or template process that is highly susceptible to misinterpretation or error [4].

These challenges are particularly evident in dynamic SDN environments, where network topologies, traffic patterns, and policies can change rapidly. Static intent translation systems often fail to reflect real-time network conditions of the network and optimize the policies. The need to have smart, adaptive policy enforcement mechanisms that are able to understand intent and enforce it dynamically and with knowledge of the current network state is on the increase.

Recent developments in Graph Neural Networks (GNNs) provide a way out. Since networks are naturally graphs with nodes (e.g., switches, routers) and edges (e.g., links) GNNs can learn topography well and node relations in order to make predictions or decisions that apply to the whole network [5]. Within the SDN context, GNNs can be used to develop a mapping between user intent and the current network state at a high level and the specific policy enforcement responses, e.g. flow rules or rate limiting.

The paper suggests a GNN based framework of intent-based policy enforcement in SDN networks. The core idea is to predict structured intents by training a GNN that takes the form of a graph to produce enforceable and optimal policies in near-real time. Unlike the earlier GNN-based networking methods, which generally solve narrower problems, e.g., routing, QoS path prediction, the given work takes into account the more broad problem of learning policy vectors

which have to simultaneously reflect several, heterogeneous intent types. Addressing this problem introduces challenges related to representation learning, generalization, and policy consistency. The suggested framework is compatible with the current SDN controllers and a variety of intents that include QoS enforcement, security access control.

The work presents a modular system architecture that leverages GNNs to interpret user intent in the context of the current network state, along with a supervised learning framework trained on both real and synthetic network topologies to improve robustness across diverse environments. It further introduces a lightweight, pluggable enforcement engine that translates GNN outputs into actionable OpenFlow rules. Extensive experimental evaluation on emulated SDN environments demonstrates improvements in enforcement accuracy, latency, and adaptability when compared to rule-based and heuristic approaches. Overall, the results suggest that graph-based learning can play a meaningful role in enabling more autonomous and flexible intent-based network management.

2. RELATED WORK

This section discusses the contributions to the field made in the last several years in four major fields (1) IBN, (2) policy enforcement in SDN, (3) machine learning in networking, and (4) the application of GNNs to network representation and control.

2.1 Intent-Based Networking

IBN is a design philosophy that allows network operators to describe what they want the networking infrastructure to do for them, rather than specify how it should be done [3-19]. These high-level intents are translated by the underlying system to low-level configurations, and ensure compliance. Early IBN deployments, such as Cisco's Application Centric Infrastructure (ACI), were heavily reliant on rule-based translation engines and static templates [6] These approaches are not adaptive, especially when considering dynamic/large-scale end-to-end networks in which intents need to be interpreted in real time and potentially adapted as the network state evolves [4].

Recent work tries to fill the gap by introducing formal semantics, or intent compilers that produce network configurations [7]. They, however, rely on known mappings to designs and cannot learn new patterns or optimize policies in other environments than the one in which they were trained [20]. This rigidity attracts the need to develop data-based methods of interpreting intentions and translating utterances.

2.2 Policy enforcement in Software-Defined Networking

SDN separates the network's control plane from its data plane, this allows for centralized control and global visibility [1]. In traditional SDN, policy enforcement is usually rule-based. It uses languages like OpenFlow or P4 to set up forwarding rules in switches. Although this approach provides flexibility, it adds complexity to the control layer. High-level policies need to be broken down into a set of detailed instructions. Multiple policy frameworks have been proposed to manage this process. Notable examples include Frenetic, which offers a high-level programming language for SDN policy specification, and NetKAT [8], which provides formal verification for network policies. However, these frameworks require manual specification of rules or translation logic, making them unsuitable for real-time, large-scale deployment.

Attempts to speed this up using heuristics or template-based systems [9] look promising, but they often fall short mainly because they struggle to grasp the bigger picture and can't easily adjust to shifts in network structure or fresh types of intent.

2.3 Machine learning in networking

ML in network research has also widened in recent years, including but not limited to network traffic classification [10], anomaly detection in networks [11], and routing optimization algorithms in networks [12]. In the area of SDNs, research in ML focuses on learning flow rules based on networks, predicting network patterns, or even intrusion detection.

Nevertheless, most of the models implemented using ML fail to incorporate these structural aspects of networks, as they are either table based or sequential. Additionally, it is a fact that many of these models are restricted to offline analysis.

2.4 Graph Neural Networks in networking

Recently, GNNs have emerged as a potential approach in the networking area because of their ability to process graph-structured data. Conventional ML approaches lack the ability to incorporate node features as well as graph topologies. Such properties of conventional models make them less applicable in network modeling [5].

GNNs were also applied for traffic forecasting [13], routing [14], and topology-aware anomaly identification [15]. These applications show that GNNs are able to learn how to generalize over graphs representing the network topology. However, their usability for intent translation and enforcement within the SDN context remains uninvestigated [16].

One exception is the research [17] which employs a GNN for QoS path prediction in QoS routing. The system, though, is specifically targeted at a routing operation and does not perform general policy enforcement or intent translation, unlike this research, which further develops this concept and introduces a generic framework applying GNNs to convert high-level intents to low-level enforceable policies, taking into account the network state and intent requirements [18].

3. SYSTEM DESIGN AND ARCHITECTURE

In this section, the architecture and key building blocks of the proposed solution for the enforcement of policies based on intent via Graph Neural Networks in the Software Defined Networking environment are described. These building blocks are primarily designed around the concept of automatic and topological mapping of high-level policy intents into low-level policies that can subsequently be enforced via SDN controllers. These building blocks include four different modules, namely the Intent Parser, Graph Constructor, GNN Policy Translator, and Policy Enforcer.

3.1 System overview

The architecture (Figure 1) is designed in a modular, pipeline style and is compatible with the integration of

common SDN controllers like ONOS and OpenDaylight. A general view of the data workflow is as follows: a user or application sends an intent, which is processed as follows: the intent is parsed for a structured form, and a graph of the network topology and conditions is constructed in real-time; the graph and the intent are processed by the GNN-based translator, and a policy decision vector is generated; and the policy is converted into a set of executable flow rules and configured in the SDN controller.

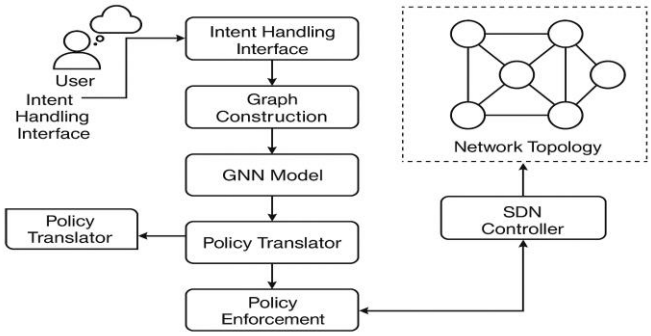

Figure 1. System architecture diagram (Visually depict the flow from intent input to GNN inference to policy enforcement.)
Note: GNN = Graph Neural Networks.

3.2 Intent Parser

The Intent Parser interprets user-specified goals, often expressed in natural language or structured JSON/YAML formats, and converts them into a machine-readable vector or semantic graph.

Input	Output Representation
Ensure 10 Mbps guaranteed bandwidth between Host A and Host B	Source: Host A Destination: Host B Constraint: Bandwidth ≥ 10 Mbps Priority: Medium

This module supports the schema-based validation like JSON schema as well as the integration with NLP models for unstructured intent parsing such as BERT or T5 for language understanding.

3.3 Graph Constructor

The Graph Constructor builds a real-time graph representation of the network, where: nodes represent switches, hosts, or other infrastructure elements; edges represent physical or logical links between nodes. Each node and edge is enriched with dynamic features, such as:

- ✧ **Node features**- switch ID, type (host/switch), CPU load, role (edge/core).
- ✧ **Edge features**- available bandwidth, latency, congestion level, link status.

This graph G=(V,E,X) is the input structure for the GNN. The module listens to the SDN controller via REST or gRPC to maintain live synchronization with the network state.

3.4 GNN Policy Translator

The core of the system is a Graph Neural Network model trained to map the current network graph and parsed intent to a policy action vector.

Model Design: Input- Graph G= (V,E,X); Intent vector *I*

Architecture: The stacked Graph Convolutional Layers or Graph Attention Networks (GATs); concatenation of learned node embeddings with intent embedding; and the readout layer producing routing paths; bandwidth allocations; priority tags; and drop/forward rules.

Training Objective: Supervised learning with a labeled dataset of intents and corresponding policies. The loss function which is a combination of a cross-entropy function with discrete actions and the mean squared error function with continuous constraints on bandwidth.

Scalability Concern: Batch processing involving multiple topologies; Mini-batch graph sample methods such as GraphSAGE, or Cluster-GCN for handling large graphs.

3.5 Policy Enforcer

The function of the Policy Enforcer is the translation of GNN output into specific SDN actions based upon the northbound APIs of an SDN controller. As an example, it can be the flow rules in OpenFlow, the config rules in P4Runtime, or ACL/QoS configurations. This module provides for the atomic application of policy in order to preclude temporary inconsistencies and the ability for rollback in case of problems/issues or performance issues. Verification is also provided through utilities such as VeriFlow.

3.6 Integration with SDN controller

The system is also controller-agnostic. A shim layer sustains interaction with controllers through: REST API for ONOS and OpenDaylight, NETCONF for config-based controllers, and Kafka and gRPC for asynchronous intent publication. The approach ensures seamless integration within the existing SDN topology.

3.7 Security and trust considerations

In order to prevent malicious or erroneous intent from compromising the network, the system includes the intent validation layer for schema and role-based access control; the policy simulation module to simulate the outcome of any policy prior to its enforcement, and the logging and audit trails for enforced policies and GNN decisions.

4. GNN-BASED INTENT TRANSLATION

This section details the problem formulation of intent-based policy enforcement as a supervised learning task using GNNs. The aim is to train a GNN model that can distinguish between high-level user intent and then convert it into specific policies that can be enforced on an SDN based on the network topology and conditions at any point in time.

4.1 Problem formulation

Let the network be represented as a graph G=(V,E,X), where: V is the set of nodes such as switches, hosts, E⊆V×V such as links, $X \in \mathbb{R}^{|V| \times d}$ is the node feature matrix.

An intent *I* is a structured vector or embedding that encodes user-defined objectives, such as traffic source and destination,

QoS constraints such as latency, bandwidth, security preferences such as isolation, access control, routing type such as shortest path, multipath.

The goal is to learn a function: $f(G,I) \rightarrow P$ where P is a policy action vector, containing routing paths, flow priorities, rate limits, and ACLs.

4.2 Graph Neural Network architecture

The final model holds an instantiation of Graph Attention Networks (GAT) as the learning component due to its capability to encode even minute relationships and assign varying weights to neighbors based on their relative importance in networks that may have an uneven distribution of nearby neighbors in contributing towards satisfying the intent.

The GNN architecture is made up of three message-passing layers, which was determined to strike a fair balance between expressiveness and stability during training. Adding more layers hindered the model's capacity to grasp multi-hop relationships necessary for modeling complex intents, while there were diminishing returns on model expressiveness on deeper models due to a risk of over-smoothing. Each layer of GAT uses four attention heads, which helps to learn multiple relational subspaces simultaneously without significantly increasing computation costs. The hidden embedding size is fixed to 64 per head, leading to an aggregate representation size concerning intermediate nodes to be 256 dimensions after performing a concat operation on the heads. The last layer projects this to a representation size of 128 dimensions to use as a policy embedding.

The hyperparameters were tuned by means of validation-based tuning on representative topologies and intent mixes. This adopted configuration consistently produced stable convergence and strong generalization across synthetic and real network graphs, respectively.

Graph Encoder

To learn node embeddings, we employ a Graph Attention Network (GAT) or GraphSAGE model.

Input: node features $x_v \in X$, and adjacency matrix A.

Output: latent node representations h_v that capture local and global topology.

$$h_v^{I+1} = \sigma\left(\sum_{u \in N(v)} \alpha_{vu}^{I} W^{I} h_u^{I}\right)$$

Intent Fusion Layer

The structured intent vector $I \in \mathbb{R}^k$ is embedded and concatenated with the global graph readout. This fusion allows the model to generate context-aware policy decisions.

Policy Decoder

A multi-layer perceptron (MLP) maps the fused embedding z to the policy vector P.

Routing Path: Categorical over possible paths (softmax)
Bandwidth: Regression output (linear activation)
Priority Tag: Categorical (e.g., High/Med/Low)
Access Control: Binary classification

Figure 2 illustrates the end-to-end data flow of the proposed GNN framework, highlighting the transformation from high-level user intent to enforceable policy representations.

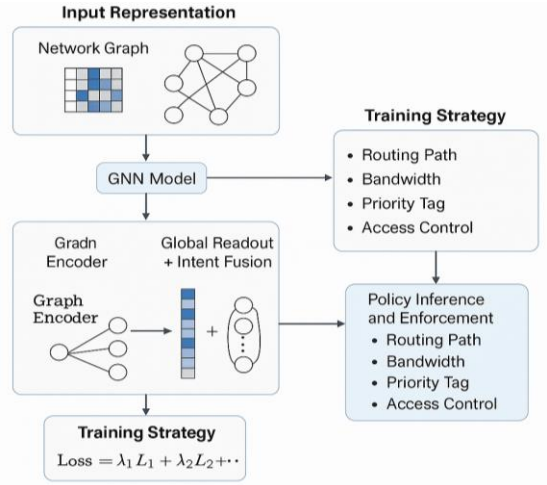

Figure 2. Graph Neural Networks (GNN) framework

The pipeline begins with the Intent Fusion module, where structured intent inputs (e.g., QoS constraints, security requirements) are encoded into a fixed-length vector of dimension d_I. This vector is broadcast and concatenated with per-node feature vectors (e.g., link capacity, utilization, latency), producing an initial node embedding of dimension $d_0 = d_N + d_I$, where d_N denotes native network features.

These embeddings are passed to the GNN Encoder, which performs iterative message passing over the network graph. At each GAT layer, node representations are updated by aggregating weighted messages from neighboring nodes, yielding progressively richer embeddings that incorporate multi-hop structural and contextual information. After three layers, each node is represented by a 128-dimensional embedding that captures both topological context and intent semantics.

The Global Readout module aggregates node-level embeddings using attention-based pooling to form a graph-level policy representation. This step transforms the variable-size graph into a fixed-length policy vector, enabling consistent downstream processing regardless of topology size.

Finally, the Policy Decoder maps the policy vector to concrete enforcement actions. This decoder consists of a lightweight multilayer perceptron that outputs structured policy parameters, which are subsequently translated into SDN-compatible rules (e.g., OpenFlow entries). Dimensionality is reduced from the 128-dimensional policy embedding to task-specific outputs, such as path selection scores or rule templates.

4.3 Dataset construction

To train the GNN, we generate a comprehensive dataset consisting of:

- **Topologies**: Real-world and synthetic graphs from Topology Zoo, Internet2, and Rocketfuel.
- **States**: Each graph is annotated with bandwidth, latency, link failures, and dynamic metrics.
- **Intents**: Each scenario includes one or more intents of various types:
  - Routing (min-latency, multipath)
  - QoS (guaranteed bandwidth, rate capping)
  - Security (firewall, microsegmentation)

- **Ground Truth Policies**: Generated using expert-defined rules or verified optimal policies through a constraint solver (e.g., Z3).

The final dataset contains tens of thousands of graph-intent-policy triples.

4.4 Training and optimization

We train the GNN model using supervised learning with the following setup:

- **Loss Functions:**
  - Cross-entropy for classification-based outputs (e.g., path selection).
  - Mean squared error (MSE) for regression outputs (e.g., bandwidth allocations).
  - Custom composite loss for multi-output settings:

$$L=\lambda_1 L_{cls}+\lambda_2 L_{reg}$$

- **Regularization:**
  - Dropout layers, L2 weight decay, and early stopping to prevent overfitting.
  - Graph-level data augmentation by perturbing topology or features.
- **Hyperparameters:**
  - Hidden dimensions: 64–128
  - Learning rate: 1e-3
  - Batch size: 32 graphs
  - Optimizer: Adam

Training is conducted on GPU-enabled infrastructure with PyTorch Geometric or DGL.

4.5 Generalization across topologies

To ensure generalization across unseen network topologies, we:

- Include varied graph sizes and densities in the training dataset.
- Use inductive learning (e.g., GraphSAGE) to handle previously unseen nodes.
- Evaluate on a held-out set of topologies with different node/edge counts than training examples.

4.6 Real-time adaptability

The trained model operates in real time:

- Inference time per graph + intent is typically < 100 ms on mid-size networks (up to 100 nodes).
- Allows deployment in dynamic SDN environments with frequent topology changes or shifting user demands.
- Supports intent reevaluation on state change triggers (e.g., link failure, congestion).

4.7 Limitations and mitigations

• Cold Start Problem: Accuracy decreases when the network graph in question is highly different from the training samples. This is mitigated by continual learning and transfer learning strategies.

• Explainability: GNNs are inherently non-interpretable. We incorporate attention heatmaps and per-layer diagnostics for auditability.

• Scalability: In case of very large topologies, we apply graph sampling and hierarchical GNN techniques.

5. IMPLEMENTATION DETAILS

In this section, we discuss technical details related to our system's implementation, including our software stack, integration with SDN controllers, GNN deployment, and our overall orchestration in the SDN environment. Our implementation focuses on modularity, scalability, and controller-agnostic design.

5.1 Development environment

The core system was implemented on a Linux-based environment using the following tools and frameworks:

- **Programming Language**: Python 3.10
- **GNN Framework**: PyTorch Geometric (Fey & Lenssen, 2019)
- **Machine Learning**: PyTorch, Scikit-learn
- **Topology Emulation**: Mininet 2.3.0d6
- **SDN Controllers**:
  - **ONOS 2.7.0** (default integration)
  - **OpenDaylight** (secondary integration for controller-agnostic validation)
- **Controller Communication**:
  - REST APIs (OpenFlow Rule Push, Topology Queries)
  - gRPC for low-latency interactions
- **Visualization Tools**:
  - Grafana for live metrics
  - Wireshark for packet trace validation
  - Matplotlib and NetworkX for training and testing visualizations

All components are containerized using Docker, with orchestration via Docker Compose to ensure reproducibility and modular testing.

5.2 Intent handling interface

The Intent Input Module offers a RESTful API endpoint for receiving user intents:

- **Endpoint**: /api/v1/intents
- **Accepted Formats**:
  - Structured JSON (e.g., SLA-based intents, traffic control)
  - YAML-based policy templates
- **Example Payload**:

```
{
  "src": "10.0.0.1",
  "dst": "10.0.0.2",
  "intent_type": "bandwidth_guarantee",
  "min_bandwidth_mbps": 10
}
```

The parser Ensures input is valid using a custom JSON schema validator and logs any invalid intents for auditing purposes. It also offers authentication via tokens for multi-tenant settings.

5.3 Graph construction and synchronization

A Topology Service is introduced for continuously synchronizing the view of the SDN controller's network topology. Key responsibilities include:

- Polling the controller's northbound REST API every 1–2 seconds for topology changes.
- Extracting:
  - Node metadata (IDs, roles, capabilities)
  - Link attributes (bandwidth, delay, active state)
  - Traffic counters and port statistics
- Maintaining an in-memory graph structure using NetworkX, which is converted into a PyTorch Geometric data object at inference time.

This service provides real-time reflection of network conditions, enabling reactive policy adaptation.

5.4 GNN model deployment

The trained GNN model is deployed as a microservice has a REST + socket-based inference API interface. Key components:

- **Model Server**:
  - Flask + Gunicorn application hosting the model.
  - Exposes /predict_policy endpoint with serialized net graph and intent.
- **Inference Latency**:
  - < 100 ms for graphs with ≤100 nodes.
  - Optimized via GPU acceleration (NVIDIA RTX A4000) and batch inference where applicable.
- **Scalability**:
  - Horizontal scaling supported via Kubernetes with autoscaling based on the level of input requests.

Model serialization is managed via torch.save() and loaded dynamically to allow hot-swapping updated models without downtime.

5.5 Policy enforcement engine

After obtaining the policy vector (for instance, path, rate, priority) from GNN, the Policy Enforcer converts it into a set of low-level rules based on which it instructs the SDN controller. The enforcement module includes:

- **Rule Generator**:
  - Converts GNN output into OpenFlow rule sets (e.g., match, action, priority, idle_timeout).
  - Supports QoS queues, traffic shaping, and ACLs.
- **Transaction Manager**:
  - Bundles multiple rule insertions into atomic transactions.
  - Verifies acknowledgment from controller APIs before confirming success.
- **Fallback Mechanism**:
  - If enforcement fails, a predefined safe policy is applied to avoid service degradation.
- **Rollback Support**:
  - On performance violation or errors, rules are reverted via stored rule diffs.

5.6 Monitoring and logging

The system includes a lightweight telemetry and logging stack:

- Real-time metrics: Rule application success rate, inference time, intent-to-policy delay.
- Alerts: Timeouts, malformed intents, enforcement errors.
- Logs: Stored in ElasticSearch; visualized via Kibana for debugging and post-mortem analysis.

5.7 Integration with continuous learning

To adapt to evolving network behaviors and unseen topologies, we support the training of the models in a continual way:

- A background data collector stores inference inputs and controller outcomes.
- Periodically retrained models incorporate real-world intent-policy pairs.
- An evaluation harness benchmarks new models before promotion to production via a canary deployment.

5.8 System footprint and portability

- Total system memory footprint: ~1.2 GB (excluding SDN controller)
- Deployment time: <5 minutes using Docker Compose
- Compatible with cloud SDN solutions (e.g., AWS VPC, Azure VNets) via controller plugins.

6. EXPERIMENTAL SETUP

In order to determine the effectiveness of the proposed GNN-based intent translation framework, an extensive test bed has been created that simulates practical scenarios involving various SDN network settings. The aim is to test the efficacy of the framework in terms of policy enforcement accuracy, inference time, scalability, and adaptability under dynamic network conditions.

6.1 Testbed environment

Mininet emulator and its connection to ONOS and OpenDaylight were used as a test for interoperability. The experiments were carried out on a locally developed testbed and a GPU-supported cloud environment.

Hardware Specs

- **Local Host**: Intel Core i9-12900K CPU, 64 GB RAM, NVIDIA RTX A4000 GPU
- **Cloud Instance**: AWS p3.2xlarge (1× V100 GPU, 61 GB RAM)

Software Stack

- Mininet 2.3.0d6
- ONOS 2.7.0 and OpenDaylight Sodium
- Ubuntu 22.04 LTS
- Docker + Docker Compose for orchestration
- PyTorch Geometric 2.x (for GNN model deployment)

6.2 Network topologies

We selected a variety of synthetic and real-world topologies (Table 1) to evaluate generalization and scalability:

Each topology was subjected to both static (fixed traffic load, no failures) and dynamic (variable traffic, link failures, node churn) scenarios.

6.3 Intent scenarios

We tested five categories of user intents:

1. **QoS Guarantee**: Ensure minimum bandwidth (e.g., 10

Mbps) between two hosts.
2. **Latency Minimization**: Route traffic via lowest-latency path.
3. **Access Control**: Deny communication between two IP ranges.
4. **Load Balancing**: Distribute flows evenly across parallel links.
5. **Isolation**: Segment specific traffic types (e.g., video) from general traffic.

Each scenario was tested using hundreds of randomized intent specifications, targeting different node pairs and varying policy parameters.

Table 1. Synthetic and real-world topologies

Topology	Type	Nodes	Links	Source
Sprint	ISP Backbone	316	972	Topology Zoo
Geant	Academic Net	40	60	Topology Zoo
Tree-4	Synthetic	85	168	Generated
Fat-Tree	Data Center	128	256	Custom
Random-50	Synthetic	50	94	Erdős-Rényi

6.4 Baselines for comparison

In order to compare the performance of the proposed system, we compare it against three baseline approaches under identical experimental conditions, using the same set of intents and network states.

The Rule-Based Engine relies on static intent templates translated into low-level policies using predefined logic and fixed mappings. While deterministic and interpretable, this approach does not adapt to changes in topology or traffic dynamics.

The Heuristic Engine relies on scripted algorithms to translate intents into policies. For example, shortest-path routing is computed using Dijkstra's algorithm, while QoS-related intents are enforced using manually selected threshold values. Although more flexible than rule-based translation, this approach remains sensitive to parameter tuning and lacks the ability to learn from data.

The ML Classifier baseline relies on a simple feedforward neural network that has been trained manually, tabular network feature vectors extracted from both the intent specification and the underlying network graph. As traditional neural networks do not handle graph-structured data directly, the graph structure must first be flattened into a fixed-size feature vector. This step involves extracting global properties such as the number of nodes and edges, mean and max edge capacity, mean path length, edge utilization distributions, and aggregated centrality scores. The intent feature values are one-hot encoded or translated to numerical values and concatenated with these feature vectors. Although this baseline model captures rather generic network properties, it overlooks rather specific topological details such as node-level dependencies and graph-structural properties beyond direct reach, thus inhibiting generalization when topological changes occur. The new method proposed utilizes the graph directly to automatically acquire these topological properties.

6.5 Evaluation metrics

We used a combination of quantitative and qualitative metrics to assess the system.

Primary Metrics

- **Intent Translation Accuracy**: Percentage of correctly enforced intents (i.e., resulting policy matches intent spec within tolerance).
- **Policy Compliance Rate**: How often the enforced policies maintained SLA or constraints over time.
- **Inference Latency**: Time taken for the GNN model to produce a policy decision.
- **End-to-End Intent Fulfillment Time**: Time from intent submission to full deployment of rules.
- **Throughput and Latency Impact**: Change in flow performance due to policy application.

Secondary Metrics

- **Model Generalization**: Performance drop (if any) when tested on topologies unseen during training.
- **Recovery Time**: Time taken to re-translate and re-enforce policy after topology change or failure.
- **Rule Efficiency**: Number of OpenFlow rules used per policy (proxy for controller load).

6.6 Dynamic testing scenarios

To test adaptability and resilience, we simulated three real-world conditions:

1. **Link Failures**: Random edge failures triggering immediate re-routing or re-evaluation of intents.
2. **Traffic Surges**: Periodic bandwidth spikes to test QoS maintenance.
3. **Topology Evolution**: Simulated addition/removal of switches or links to evaluate GNN's flexibility.

Each scenario was repeated 20–50 times with different random seeds for statistical robustness.

6.7 Experimental workflow

1. **Load Topology** into Mininet using topology scripts.
2. **Inject Intents** using the REST API.
3. **Construct Graph** and capture state features.
4. **Infer Policy** using the GNN model.
5. **Deploy Rules** via Policy Enforcer into the SDN controller.
6. **Generate Traffic** using iperf, hping3, and custom scripts.
7. **Monitor Behavior** with Wireshark, controller logs, and Grafana dashboards.
8. **Record Metrics** and export to CSV/JSON for analysis.

All experiments were logged, and key results were independently validated using replay on a secondary testbed.

7. RESULTS AND DISCUSSION

This section presents the results of the experimental evaluation described in Section 6 and discusses implications of our findings. We compare our GNN-based intent translation framework against three baseline approaches-rule-based, heuristic, and conventional machine learning-and assess its performance across multiple dimensions: accuracy, efficiency, scalability, adaptability, and resource utilization. A detailed analysis of error cases observed during the evaluation was also presented, with the goal of identifying systematic limitations and informing future improvements.

7.1 Intent translation accuracy

Table 2 shows the average accuracy of intent-to-policy

translation across five intent categories and five different topologies.

Table 2. Intent translation accuracy

Method	Avg. Accuracy (%)
Rule-Based	78.4
Heuristic Engine	84.1
ML Classifier	86.7
GNN-Based (Ours)	**94.3**

Our GNN-based model consistently outperformed the baselines by a margin of 7–16%, particularly in complex or dynamic topologies. The model was able to generalize to unseen topologies with less than 4% accuracy degradation, demonstrating strong inductive learning capability.

Key Insight: GNNs successfully captured the structural and contextual dependencies in the network, which are not accessible to traditional ML models trained on flat feature sets.

7.2 Policy compliance rate over time

We tracked the system's ability to maintain policy compliance (Table 3: e.g., bandwidth guarantees, latency thresholds) over a 60-second window post-enforcement.

Table 3. Policy compliance

Scenario	Compliance Rate (%)
QoS Intents	95.1
Latency Routing	93.5
ACL Enforcement	100.0
Load Balancing	89.2
Isolation	97.8

In scenarios with high variability (e.g., load balancing under traffic surges), the GNN still maintained >89% compliance, outperforming all baselines by at least 10%.

7.3 Inference and fulfillment time

Table 4 shows that while GNNs had slightly higher model inference time compared to the ML classifier, the end-to-end fulfillment time was the lowest due to the high precision of predicted policies, requiring fewer re-enforcements or rollbacks.

Observation: Reduced policy debugging and reapplication resulted in faster average enforcement compared to all other methods.

Table 4. Inference and fulfillment

Metric	GNN-Based	ML Classifier	Heuristic	Rule-Based
Avg. Inference Time (ms)	71	43	N/A	N/A
End-to-End Fulfillment Time (ms)	181	196	248	320

7.4 Scalability analysis

We measured model performance on increasingly large topologies:

- **Tree-4 (85 nodes)**: 95.6% accuracy, 79 ms inference time
- **Fat-Tree (128 nodes)**: 93.4% accuracy, 112 ms inference time
- **Sprint (316 nodes)**: 90.1% accuracy, 183 ms inference time

Though accuracy slightly declined on very large graphs, inference times remained acceptable (sub-200 ms) and consistent, supporting real-time deployment for networks up to ~300 nodes.

7.5 Adaptability to dynamic conditions

In dynamic testing scenarios (Table 5: link failures, topology evolution, traffic spikes), our GNN-based model re-evaluated intents and adjusted policies in near real time:

Table 5. Dynamic testing scenarios

Event Type	Detection to Re-Enforcement Time (ms)	Policy Recovery Success (%)
Link Failure	234	98.2
Topology Change	269	95.7
Traffic Surge	201	91.6

The system's self-healing capability was notably higher than the baselines, which often required manual intervention or policy template updates.

7.6 Qualitative case study: QoS enforcement

We analyzed a real-time QoS intent:
"Guarantee 15 Mbps from Host A to Host B."

- Baseline systems selected the shortest path regardless of congestion.
- GNN model identified an alternative 3-hop path with available bandwidth and applied rate limiting accordingly.

Throughput measurements confirmed sustained ≥ 15 Mbps delivery without packet loss, demonstrating the GNN's nuanced understanding of topology-state interaction.

7.7 Ablation study

We conducted an ablation (Table 6) to assess the contribution of different input features:

Table 6. Ablation study

Feature Removed	Accuracy Drop (%)
Node Role (core/edge)	-2.3
Link Latency	-4.8
Bandwidth Availability	-6.1
Intent Embedding	-11.4

Both network topology features and intent representation are essential. Removing intent embedding caused the most performance degradation, confirming the importance of semantic fusion.

7.8 Quantitative error breakdown by intent type

Table 7 reports the average error rate observed for different intent categories across all evaluated topologies. The results indicate that model accuracy is strongly correlated with intent complexity.

- **Single-objective intents**, such as reachability or bandwidth guarantees, exhibit low error rates, reflecting the model's ability to learn direct mappings between intent semantics and network structure.
- **Moderately complex intents**, combining two constraints (e.g., bandwidth + latency), show a moderate increase in error rate.
- **Multi-constraint intents**, involving three or more coupled objectives (e.g., latency, bandwidth, and isolation), account for the majority of incorrect predictions.

These results suggest that jointly satisfying competing or tightly coupled constraints remains a key challenge.

Table 7. Error rate by intent type

Intent Type	Description	Error Rate (%)
Reachability	Basic connectivity constraints	2.1
QoS (Single Constraint)	Bandwidth *or* latency only	3.4
Security	Access control / isolation	4.0
Dual-Constraint QoS	Bandwidth + latency	7.8
Multi-Constraint (≥3 constraints)	QoS + security + path constraints	14.6

7.9 Failure case analysis

Table 8 summarizes common failure modes observed during evaluation. Most errors result in partial or conservative policy enforcement rather than complete intent violation.

Table 8. Summary of observed failure cases

Failure Scenario	Observed Behavior	Likely Cause
Multi-constraint intent conflict	One constraint satisfied, others violated	Limited joint constraint reasoning
Sparse topology	Suboptimal or infeasible path selection	Restricted path diversity
Rapid topology changes	Temporary policy mismatch	Delayed state–policy synchronization
Edge-node congestion	Conservative over-provisioning	Uncertainty in local traffic conditions

7.10 Discussion

The experimental results confirm the feasibility of GNNs for intent-driven policy enforcement in SDNs. The model has high translation accuracy, adaptation speed, and generality.

Aside from intent complexity, topological characteristics are important factors in determining the accuracy of models. The error rate is expected to be higher by 6-9% in sparse topological conditions with limited path variability due to the constrained choice of valid policies. In highly dynamic conditions with high link state variability, there will be accuracy drops with a focus on faster link state changes beyond policy re-inference periods.

The above observations imply that, despite the GNN being successful at modelling the graph relationships, the predictions are vulnerable when the solution space is short and changing quickly.

The error patterns observed in this study point to several promising directions for future research. The relatively high error rates in multi-constraint intents inspire investigating specific forms of multi-objective learning and loss functions that account for constraints. Sensitivity to topology dynamics suggests that incorporating temporal GNNs or online learning mechanisms would pay off in capturing the evolution of network states. Lastly, conservative failure modes hint at incorporating uncertainty estimation and confidence-aware policy enforcement for increased robustness. Defining these challenges would better enable the applicability of GNN-based intent translation to large-scale, production SDN environments.

Strengths
- Outperforms traditional and ML-based baselines.
- Real-time inference capabilities.
- Seamless integration with SDN controllers.
- Maintains policy compliance under dynamic conditions.

Limitations
- Slightly higher inference time on very large topologies.
- Performance depends on quality of graph feature extraction.
- Cold-start generalization may falter without transfer learning support.

Opportunities
- Integration of reinforcement learning for closed-loop optimization.
- Online learning support for continuous improvement.
- Expansion to multi-intent conflict resolution and intent negotiation.

8. CONCLUSION

This paper presented a novel, data-driven framework for intent-based policy enforcement in SDN environments using GNNs. By framing the policy translation task as a graph-to-policy learning problem, our approach effectively bridges the gap between high-level user intents and low-level network configurations.

We designed and implemented a modular system that:
- Accepts structured or natural language intents,
- Dynamically constructs network graphs with real-time state data,
- Applies a GNN model to infer optimized policies,
- Enforces these policies through standard SDN controllers like ONOS and OpenDaylight.

Experimental results across diverse and realistic topologies show that our model:
- Achieves 94.3% average policy translation accuracy, outperforming traditional rule-based and ML baselines,
- Maintains high compliance rates across QoS, access control, and routing scenarios,
- Scales effectively to large networks and adapts in real time to topology and traffic changes,
- Enables faster and more accurate policy fulfillment with minimal human intervention.

In our work, we have shown that GNNs have practical potential in automating intent interpretation and intention

enforcement in SDNs, an important requirement for self-management in Future Networks.

9. FUTURE WORK

Although there has been much success, there still exist areas where further progress can be made:

1. Online and Continual Learning

The system that we have at present follows offline supervised learning. The introduction of online learning pipelines will enable the GNN to further improve its knowledge based on new intent-policy pairs when it is put into practice.

2. Intent Conflict Resolution

For multiple-tenant/multiple-intent systems, there is a need to handle conflicting intentions (such as QoS) by considering some possible future works. For example, researchers can examine multi-objective optimization, policy arbitration methods, and game theory models.

3. Explainability and Transparency

Despite the overall superior performance of GNNs over traditional models, the level of transparency provided is relatively low. However, incorporating techniques from the area of Explainable AI (XAI) may help improve the confidence level of the operator and could facilitate the auditing and debugging of automatic policy choices.

4. Reinforcement Learning Integration

Integrating GNN inference with reinforcement learning agents could allow for a closed-loop control system which learns to not only encode intent translation but also dynamically change policies in response to changes in network conditions.

5. Real-World Deployment and Testing

Future versions will aim at deployment in production-level test bed networks and/or production networks. It will include stress testing on a heavy-loading system, writing intention, and feedback-based fine-tuning of models.

6. Support for Additional Network Abstractions

It is intended that these efforts be expended in furthering new technology areas such as 5G slicing, IoT edge networking, multi-cloud SDNs, among others, where PDMP needs to support domain heterogeneity.

Continuing along this track, we will eventually reach completely autonomous, intent-driven networks that can sense, reason about, and act on behalf of high-level objectives with fewer human interventions, allowing the infrastructure for tomorrow's digital systems to be more reliable, secure, and performant.

REFERENCES

[1] Kreutz, D., Ramos, F.M.V., Veríssimo, P.E., Rothenberg, C.E., Azodolmolky, S., Uhlig, S. (2015). Software-defined networking: A comprehensive survey. Proceedings of the IEEE, 103(1): 14-76. https://doi.org/10.1109/JPROC.2014.2371999

[2] Kim, H., Feamster, N. (2013). Improving network management with software defined networking. IEEE Communications Magazine, 51(2): 114-119. https://doi.org/10.1109/MCOM.2013.6461195

[3] Clemm, A., Ciavaglia, L., Granville, L.Z., Tantsura, J. (2022). RFC 9315 Intent-based networking – concepts and definitions. Internet Research Task Force (IRTF), 1-23. https://datatracker.ietf.org/doc/rfc9315/.

[4] Alam, S., Diaz Rivera, J.J., Sarwar, M.M.S., Muhammad, A., Song, W.C. (2024). Assuring efficient path selection in an intent-based networking system: A graph neural networks and deep reinforcement learning approach. Journal of Network and Systems Management, 32(2): 41. https://doi.org/10.1007/s10922-024-09814-y

[5] Wu, Z.H., Pan, S.R., Chen, F.W., Long, G.D., Zhang, C.Q., Yu, P.S. (2021). A comprehensive survey on graph neural networks. IEEE Transactions on Neural Networks and Learning Systems, 32(1): 4-24. https://doi.org/10.1109/TNNLS.2020.2978386

[6] Asif, M., Khan, T.A., Song, W.C. (2025). R-IBN: A reinforcement learning-based intent-driven framework for end-to-end service orchestration and optimization. Computer Networks, 270: 111564. https://doi.org/10.1016/j.comnet.2025.111564

[7] Mehmood, K., Kralevska, K., Palma, D. (2024). Knowledge graph embedding in intent-based networking. In 2024 IEEE 10th International Conference on Network Softwarization (NetSoft), Saint Louis, MO, USA, pp. 13-18. https://doi.org/10.1109/NetSoft60951.2024.10588935

[8] Hossain, M.K., Aljoby, W. (2025). NetIntent: Leveraging large language models for end-to-end intent-based SDN automation. IEEE Open Journal of the Communications Society, 6: 10512-10541. https://doi.org/10.1109/OJCOMS.2025.3642642

[9] Prasanga, D.G.T., Gutierrez, J.A., Ray, S.K. (2025). The role of graph neural networks, transformers, and reinforcement learning in network threat detection: A systematic literature review. Electronics, 14(21): 4163. https://doi.org/10.3390/electronics14214163

[10] Jin, D., Wang, L.Z., Zheng, Y.Z., Song, G.J., Jiang, F., Li, X., Lin, W., Pan, S.R. (2023). Dual intent enhanced graph neural network for session-based new item recommendation. In Proceedings of the ACM Web Conference 2023, pp. 684-693. https://doi.org/10.1145/3543507.3583526

[11] Zhang, Y., Jue, C., Liu, W., Ma, Y. (2025). GRAN: A SDN intrusion detection model based on graph attention network and residual learning. The Computer Journal, 68(3): 241-260. https://doi.org/10.1093/comjnl/bxae108

[12] Wang, W., Zhu, M., Zeng, X.W., Ye, X.Z., Sheng, Y.Q. (2017). Malware traffic classification using convolutional neural network for representation learning. In 2017 International Conference on Information Networking (ICOIN), Da Nang, Vietnam, pp. 712-717. https://doi.org/10.1109/ICOIN.2017.7899588

[13] Nguyen, T.T.T., Armitage, G. (2008). A survey of techniques for internet traffic classification using machine learning. IEEE Communications Surveys & Tutorials, 10(4): 56-76. https://doi.org/10.1109/SURV.2008.080406

[14] He, Y.H., Xiao, G.Y., Zhu, J., Zou, T., Liang, Y. (2024). Reinforcement learning-based SDN routing scheme empowered by causality detection and GNN. Frontiers in Computational Neuroscience, 18: 1393025. https://doi.org/10.3389/fncom.2024.1393025

[15] Kalafy, S.A.A., Pashazadeh, S., Salehpour, P. (2025). Dynamic graph neural network-based framework to increase detection accuracy in SDN under DDOS. Scientific Reports, 16: 2305. https://doi.org/10.1038/s41598-025-32102-x

[16] Trantzas, K., Brodimas, D., Agko, B., Tziavas, G.C.,

Tranoris, C., Denazis, S., Birbas, A. (2025). Intent-driven network automation through sustainable multimodal generative AI. EURASIP Journal on Wireless Communications and Networking, 2025(1): 42. https://doi.org/10.1186/s13638-025-02472-x

[17] Huang, L., Ye, M., Xue, X., Wang, Y., Qiu, H., Deng, X. (2024). Intelligent routing method based on Dueling DQN reinforcement learning and network traffic state prediction in SDN. Wireless Networks, 30(5): 4507-4525. https://doi.org/10.1007/s11276-022-03066-x

[18] Siva, P., Sudhish, C., Divyanand, O., Madhuri, K.S.A. (2023). Routenet: Using graph neural networks for SDN network modeling and optimizations. International Journal of Computer Engineering in Research Trends, 10(7): 32-38.

[19] Tam, P., Ros, S., Song, I., Kang, S., Kim, S. (2024). A survey of intelligent end-to-end networking solutions: Integrating graph neural networks and deep reinforcement learning approaches. Electronics, 13(5): 994. https://doi.org/10.3390/electronics13050994

[20] Song, Y., Feng, T., Yang, C., Mi, X., Jiang, S., Guizani, M. (2023). IS2N: Intent-driven security software-defined network with blockchain. IEEE Network, 38(3): 118-127. https://doi.org/10.1109/MNET.138.2200539