Vietnam National University, Hanoi
International School
============

M.A. Thesis

DEEP NEURAL NETWORK BASED ON GRAPH-LEVEL REPRESENTATION LEARNING FOR GRAPH-AWARE APPLICATIONS

NGUYEN DUC CHINH

Field: **Master of Informatics and Computer Engineering**
Code: **8480111.01QTD**

Hanoi - 2025

Vietnam National University, Hanoi
International School
============

M.A. Thesis

DEEP NEURAL NETWORK BASED ON GRAPH-LEVEL REPRESENTATION LEARNING FOR GRAPH-AWARE APPLICATIONS

NGUYEN DUC CHINH

Field: **Master of Informatics and Computer Engineering**
Code: **8480111.01QTD**
Supervisor: **Dr. Ha Manh Hung**

Hanoi - 2025

CERTIFICATE OF ORIGINALITY

I, the undersigned, hereby certify my authority of the study project report entitled *"Deep Neural Network Based on Graph-Level representation learning for Graph-Aware application"* submitted in partial fulfillment of the requirements for the degree of Master Informatics and Computer Engineering. Except where the reference is indicated, no other person's work has been used without due acknowledgement in the text of the thesis.

Hanoi, 22 June 2025

[illegible]

Nguyen Duc Chinh

ACKNOWLEDGEMENTS

First and foremost, I would like to express my sincere gratitude to International School, Vietnam National University, Hanoi for providing me with the opportunity and necessary support to pursue my studies and complete this thesis, to the Office of Graduate Studies for their continuous guidance and assistance, and to my research lab, CoMI (Cognitive Machine Intelligence) lab, led by Dr. Ha Manh Hung, for their invaluable support and insightful discussions throughout this journey.

I am deeply grateful to all the professors and lecturers of the Master of Informatics and Computer Engineering (MICE) program for their dedication, expertise, and invaluable knowledge. Their guidance has played a crucial role in shaping my academic and professional development.

My deepest appreciation goes to my supervisor, Dr. Ha Manh Hung, for his unwavering support, insightful guidance, and invaluable advice throughout this research. His patience and encouragement have been instrumental in helping me navigate challenges and refine my work.

I would also like to extend my heartfelt thanks to the faculty members, researchers, and experts who have contributed valuable discussions and constructive feedback, which have greatly enriched this thesis.

Finally, I am profoundly grateful to my family, friends, and colleagues for their unwavering encouragement, support, and understanding throughout this journey. Their belief in me has been a constant source of motivation and strength.

Hanoi, 22 June 2025

[illegible]

Nguyen Duc Chinh

ABSTRACT

In recent years, graph-based deep learning has emerged as a crucial research direction, particularly with the development of graph convolutional networks (GCNs). These models enable the extraction of structural information from non-Euclidean data, extending the capabilities of deep neural networks (DNNs) to fields such as cybersecurity, computer vision, and social network analysis. However, most existing studies primarily focus on node-level representations while underutilizing global graph structures.

This study proposes a DNN model based on graph-level representation learning to enhance the performance of graph-aware applications. Instead of relying solely on node features, this approach captures structural information across the entire graph, improving generalization and model performance.

The objective of this research is to optimize deep learning model for graph-structured data, enabling effective processing of complex datasets. To evaluate the proposed approach, we applied it to two critical tasks:

- SQL Injection detection, where SQL queries are represented as graphs, allowing GCNs to extract deeper features.

- Hand gesture recognition using skeletal data, integrating GCNs with attention mechanisms and spatial features to enhance accuracy.

The methodology involves extended GCN architectures, combined with self-attention mechanisms to improve graph-level representation learning. Experimental results demonstrate that the proposed approach achieves SOTA performance in SQL Injection detection with an accuracy of 99%, while also attaining 99.53% accuracy in hand gesture recognition, outperforming traditional methods.

These findings highlight the potential of deep learning on graph representations in improving AI system performance across various domains. However, this study also identifies key challenges, such as the complex preprocessing required for graph data and the dependence on labeled datasets. Future research will focus on unsupervised learning for graphs and extending models to dynamic graphs.

LIST OF ABBREVIATIONS

Abbreviation	Meaning
AI	Artificial Intelligence
ML	Machine Learning
CAP	Credit Assignment Problem
ANNs	Artificial Neural Networks
GPUs	Graphics Processing Units
TPUs	Tensor Processing Units
IoT	Internet of Things
CNNs	Convolutional Neural Networks
RNNs	Recurrent Neural Networks
GANs	Generative Adversarial Networks
GNNs	Graph Neural Networks
DNNs	Deep Neural Networks
GCNs	Graph Convolutional Networks
RecGNNs	Recurrent Graph Neural Networks
ConvGNNs	Convolutional Graph Neural Networks
GAEs	Graph Autoencoders
STGNNs	Spatial-Temporal Graph Neural Networks
SVMs	Support Vector Machines
GFT	Graph Fourier Transform
IGFT	Inverse Graph Fourier Transform
ReLU	Rectified Linear Unit
SQL	Structured Query Language
GraphConv	Graph Convolution
ViLS	Vietnamese Sign Language
ASL	American Sign Language

List of Figures

List of Tables

Contents

Chapter 1

Introduction

1.1 Rationale

Over the past decade, research in deep learning has revolutionized the field of artificial intelligence (AI) and machine learning (ML), reshaping the way intelligent systems are built and deployed. As a crucial branch of machine learning, deep learning is not just a technique, but also a systematic approach to learning and representing complex data. It focuses on constructing and learning complex concepts through hierarchical approaches or stacking multiple layers of more basic concepts, thus enabling computer systems to automatically extract features and learn from data efficiently.

One of the greatest challenges in implementing machine learning models is addressing the Credit Assignment Problem (CAP). Artificial Neural Networks (ANNs) have proven to be highly effective in solving CAP, becoming an indispensable tool for deep learning implementation. ANNs, through techniques such as backpropagation and gradient optimization, allow the training of complex models with millions or even billions of parameters. This makes deep learning a powerful and widely adopted method that surpasses the limits of traditional machine learning techniques.

In recent years, the significant advancements in hardware computational capabilities, particularly Graphics Processing Units (GPUs), along with the emergence of distributed computing platforms such as TPUs, have marked a major breakthrough in deep learning research. The parallel computation capability of GPUs has greatly reduced the training time for neural models, while advanced optimization algorithms allow models to converge faster and more accurately. At the same time, the availability of vast amounts of data, from sources such as images, text, and sensor data from the internet and IoT devices, has opened up immense opportunities for developing and testing large-scale neural models.

These breakthroughs have positioned deep learning as the driving force behind advancements in AI domains such as computer vision, speech recognition, natural language processing, and recommendation systems.

End-to-end deep learning models like CNNs, RNNs, and Autoencoders have not only transformed approaches to machine learning tasks but have also laid the foundation for the development of more complex architectures such as GANs, Attention Mechanisms, and Transformers. These innovations have elevated deep learning to new heights, making it a key tool in solving complex problems in science and engineering.

Currently, non-Euclidean data, particularly graph-based data, is becoming increasingly prevalent in many real-world applications and contexts. A graph is a powerful and flexible data structure that represents objects along with their relationships. In this structure, nodes represent objects, while edges represent relationships or interactions between them. The ability to model complex systems through graphs not only aids in visualizing data but also provides an analytical framework for various domains.

Graphs can capture structural dependencies and relationships between data components, which traditional methods based on Euclidean data (such as tables or images) cannot easily accomplish. As a result, graph data has naturally emerged in many practical applications, including:

- Social Network Analysis: Graphs can represent relationships between users, such as friend networks on social media platforms, helping to analyze influence, detect communities, or suggest connections.
- Bioinformatics: In biology and medicine, graphs are used to represent protein-protein networks, gene networks, or molecular structures, aiding in a deeper understanding of biological functions and drug discovery.
- Computer Vision: Graphs help represent spatial relationships between objects in images or videos, such as in object recognition or scene classification applications.

Additionally, graphs are also used in recommendation systems, traffic network analysis, and even financial problems, where data often have complex dependencies that are difficult to model using traditional approaches.

In recent years, Graph Neural Networks (GNNs) have proven to be highly effective in leveraging the relationships between data components. Unlike traditional Euclidean data, graphs allow the representation and exploration of complex, often non-linear relationships between objects through nodes and edges. This advantage allows models to capture not only local information but also global connections, opening up significant potential for computational optimization. Particularly, GNNs enable the construction of compact yet effective models, making them suitable for practical problems with limited resource requirements.

However, several challenges remain in applying these models to real-world problems. One of the biggest issues is that most real-world data is not readily available in graph form. Instead, data is typically in tabular, time series, or image formats. Converting this data into graph representations requires careful identification of nodes, edges, and their weights. If this process is not carried out carefully, important data may be lost or relationships may be incorrectly defined, leading to a significant drop in model performance.

Furthermore, although graph models optimize computations by exploiting data structures, their practical efficiency is still not optimal for large-scale problems. Many models still require significant computational resources to handle complex data, making them difficult to apply on resource-constrained systems or in real-time applications.

Another challenge is how to design models that are both compact and efficient. In practice, problems demand models that not only achieve high accuracy but also have sizes that are appropriate for real-world environments, such as mobile devices or embedded systems. This creates an urgent need to balance model representation capabilities with the computational resources required.

Thus, while graph models hold great potential due to their ability to leverage relationships between components to optimize computations, limitations in data conversion, computational efficiency, and model design for resource-constrained environments create significant research gaps. These challenges not only hinder the widespread application of graph models but also reduce their potential in developing advanced solutions for real-world problems.

1.2 Aim and objectives of the study

Aim

The objective of this research is to develop and enhance deep learning methods based on graph-level representation learning to improve the effectiveness of graph-aware applications. Specifically, the study focuses on constructing Deep Neural Networks (DNNs) on graph represubsentations to optimize feature learning capabilities and address current limitations in graph-related problems.

Objectives

To achieve this goal, the research focuses on the following key objectives:

- Conduct a comprehensive literature review on deep learning methods for graphs, including graph-level representation learning techniques and related applications.
- Analyze current challenges in graph representation learning, particularly in terms of generalization, representation capability, and computational efficiency.
- Develop a deep learning model based on graphs to improve the quality of graph-level representations and enhance inference capabilities in graph-aware applications.
- Experiment with the proposed model on real-world datasets, evaluating its performance against existing methods, with a particular focus on its ability to capture both global and local graph structures.
- Optimize the model to reduce computational costs, improve accuracy, and enhance scalability when applied to real-world systems.

1.3 Research question

The introduction to the research topic and the definition of the problem are presented in Section 1.1, leading to the primary objectives and research questions: *"The effectiveness of graph convolution in real-world problems and how to improve graph classification based on machine learning to better address the issue of global connectivity."*

This research project aims to demonstrate the effectiveness of graph classification by investigating the application of graphs to existing problems and exploring ways to enhance the performance of graph convolution models:

- The main contribution of this research is to validate the effectiveness of graph convolution by applying it to long-standing problems that have been traditionally addressed using other methods.
- The study also investigates various approaches to improve graph convolution by enhancing the global connectivity of components within the graph network.

1.4 Methods of the study

This study employs both theoretical and experimental approaches to address research questions by constructing, analyzing, and testing models based on graph data. The research process is carried out through the following steps:

First, the study aims to demonstrate the effectiveness of the convolutional method in classical problems. Subsequently, the research delves deeper into improving the performance of graph convolutional models.

In the phase of evaluating the effectiveness of the convolutional method on classical problems, I selected a well-established problem that has been extensively studied for many years. Numerous effective methods have been proposed to address it; however, no prior work has approached this problem using the graph convolutional method.

The initial stage involves problem formulation, aiming to solve the classical problem with comparable or superior performance compared to existing methods. The next phase consists of reviewing related research and literature, analyzing the strengths and weaknesses of previous approaches, and providing an updated overview of the field. The proposed approach is developed based on graph convolutional theories, and the model is built, tested, and optimized using publicly available datasets relevant to the problem. Evaluations are conducted based on experimental results.

Each part of the study follows a rigorous structure, beginning with problem definition and an assessment of its significance. This is followed by a comprehensive literature review, theoretical foundations, the proposed methodology, dataset description, model training and optimization processes, experimental results, and an overall evaluation.

1.5 Scope of the study

This study focuses on Graph Neural Networks (GNNs), specifically Graph Convolutional Networks (GCNs). The primary objective is to demonstrate the effectiveness and accuracy of GCNs in graph classification tasks while proposing optimizations to enhance model performance. These improvements aim to optimize training and computation processes, thereby increasing accuracy, convergence speed, and generalization ability while maintaining feasibility in terms of computational resources.

Data and Data Sources

This study utilizes publicly available graph datasets widely used in the research community. These datasets include:

- Structured graphs: Predefined graph-structured data specifically designed for graph learning tasks.
- Transformed graph data: Non-graph data sources that can be represented as graphs to facilitate training.

Models and Optimization Methods

The research focuses on applying graph convolutional models, including:

- Standard GCN models: Direct application of graph convolution operations.
- Variants of GCN: Enhancements in the convolutional core or hybrid approaches integrating other methods to optimize performance.

Practical Applications and Research Context

Graph classification is a fundamental problem in graph-based machine learning with numerous practical applications across various domains. Although GCNs have been proposed for a long time, their application remains less widespread than traditional convolutional methods, mainly due to the requirement that input data must be in graph form. However, GCNs offer advantages such as computational efficiency and feasibility for deployment on hardware-constrained systems, making them promising for resource-limited environments.

Limitations in Time and Computational Resources

Due to constraints in time and computational resources, this study focuses on optimizing models for medium-scale datasets. The goal is to develop compact models that reduce training time while maintaining competitive performance compared to existing methods, thus enhancing feasibility for real-world applications.

1.6 Research Structure

This study is organized into six main chapters as follows:

- **Chapter 1: Introduction** – This chapter presents an overview of the research, including its background, research problem, objectives, research questions, approach, significance, and the overall structure of the dissertation.

- **Chapter 2: Literature review** – This chapter provides an overview of *Graph Neural Networks (GNNs)*, deep learning methods on graphs, and related prior research. The content of this chapter is divided into three main sections:
  - **Section 1:** An introduction to deep learning on graphs, including background, research motivation, and applications of GNNs.
  - **Section 2:** Definitions of key concepts related to graphs, including fundamental terms such as vertices, edges, adjacency matrix, weight matrix, etc.
  - **Section 3:** A detailed description of *Graph Convolutional Networks (GCNs)*, including mathematical formulations, operational mechanisms, and common variants.
- **Chapter 3: Study 1** – This chapter demonstrates the superior effectiveness of Graph Convolutional Networks (GCNs) when applied to an optimization problem that has not been previously addressed using this method. It details the experimental process, including the datasets used, model training strategy.
- **Chapter 4: Study 2** – This chapter focuses on enhancing the performance of convolutional models on a well-known graph-based problem by addressing limitations related to global connectivity in graphs. It also provides a detailed description of the experimental process, covering the datasets, training strategy.
- **Chapter 5: Evaluation of Results** – This chapter analyzes and synthesizes the research findings through two main sections:
  - The first section presents and analyzes the results of each study in detail.
  - The second section consolidates the obtained results, assessing the extent to which the research objectives have been achieved.
- **Chapter 6: Conclusion** – This chapter summarizes the key contributions of the research, highlights current limitations, and proposes future research directions to address existing challenges and extend the applicability of the proposed models.

1.7 Contributions

During my master's studies in the Master of Informatics and Computer Engineering (MICE) program, with the dedicated guidance and support of my professors and the CoMI (Cognitive Machine Intelligence) research lab, especially Dr. Manh-Hung

Ha, the head of CoMI lab, I conducted preliminary research that served as a solid foundation for the completion of this master's thesis.

Table 1.1: List of related studies.

Reference	**Title**	**Venue**
[1]	Towards Lightweight Model Using Non-local-based Graph Convolution Neural Network for SQL Injection Detection	The Egyptian Informatics Journal, by the Faculty of Computers and Artificial Intelligence, Cairo University (Q1) (Revision)
[2]	A New Efficient Optimized Graph Convolutional Neural Network based Multi-Head Attentative for Sign Language Recognition	Multimedia Tools and Applications, Springer (Q1) (Submitted)
[3]	RHM: Novel Graph Convolution Based on Non-Local Network for SQL Injection Identification	2024 IEEE Symposium on Industrial Electronics & Applications (ISIEA), Kuala Lumpur, Malaysia (Published)
[4]	Lightweight Graph Convolutional Network Based on Multi-Head Residual Attention for Hand Point Classification	2024 IEEE International Conference on Visual Communications and Image Processing (VCIP), Tokyo, Japan (Published)
[5]	Enhancing Physical Rehabilitation Evaluation with Temporal Graph Convolution Networks	5th International Conference on Intelligent Systems & Networks (Accepted)

During my studies and research, in addition to my primary focus on graph-based methodologies, I had the opportunity to expand my scope to deep learning and collaborate with the CoMI research lab on several publications. These experiences have not only enriched my expertise but also provided a solid foundation for the completion of this master's thesis.

Table 1.2: List of studies in the field of deep learning.

Reference	**Title**	**Venue**
[6]	Toward improving precision and complexity of transformer-based cost-sensitive learning models for plant disease detection	Frontiers in Computer Science (Q1)
[7]	Emotional Inference from Speech Signals Informed by Multiple Stream DNNs Based Non-Local Attention Mechanism	EAI Endorsed Transactions on Industrial Networks and Intelligent Systems (Q2)
[8]	Plant Pathology Identification Using Local-Global Feature Level Based On Transformer	Indonesian Journal of Electrical Engineering and Computer Science (Q3)
[9]	Low-High Feature Based Local-Global Attention for Traffic Police Action Identification	2023 Asia Meeting on Environment and Electrical Engineering (EEE-AM), Hanoi, Vietnam
[10]	You Only Look Once Based-C2fGhost Using Efficient CIoU Loss Function for Airplane Detection	2024 9th International Conference on Frontiers of Signal Processing, Paris, France
[11]	Human Detection Based Yolo Backbones-Transformer in UAVs	2023 International Conference on System Science and Engineering (ICSSE), Ho Chi Minh, Vietnam
[12]	An Effective Method for Detecting Personal Protective Equipment at Real Construction Sites Using the Improved YOLOv5s with SIoU Loss Function	2023 IEEE International Conference on Research, Innovation and Vision for the Future, Hanoi, Vietnam
[13]	VNEMOS: Vietnamese Speech Emotion Inference Using Deep Neural Networks	2024 9th International Conference on Integrated Circuits, Design, and Verification (ICDV), Hanoi, Vietnam

Chapter 2

Literature review

This chapter provides a literature review on Graph Neural Networks (GNNs), focusing on their historical development, key methodologies such as Recurrent GNNs and Convolutional GNNs, and fundamental concepts of graph structures. It aims to establish a theoretical and practical foundation, elucidating how GNNs process graph data while drawing comparisons with related techniques, including network embedding and graph kernel methods.

2.1 Background

2.1.1 History of Graph Neural Networks (GNNs)

Graph Neural Networks (GNNs) were first studied in the late 1990s. The work of Sperduti et al. (1997) [14] pioneered the application of neural networks to directed acyclic graphs, laying the foundation for subsequent research on GNNs.

The term "Graph Neural Networks" was first introduced in the study of Gori et al. (2005) [15] and later expanded by Scarselli et al. (2009) [16] and Gallicchio et al. (2010) [17]. These methods belong to the class of Recurrent Graph Neural Networks (RecGNNs), where information from neighboring nodes is iteratively propagated to learn representations of the target node. This process continues until a stable state is reached. However, this approach has significant computational costs, and many recent studies have focused on improving the efficiency of RecGNNs [18, 19].

2.1.2 Development of Convolutional Graph Neural Networks (ConvGNNs)

With the success of Convolutional Neural Networks (CNNs) in computer vision, researchers sought to extend convolution operations to graph data. This led to the emergence of Convolutional Graph Neural Networks (ConvGNNs), where convolutions are redefined to handle non-Euclidean structured data.

ConvGNNs are divided into two main approaches:

- **Spectral-based methods**: The first significant study in this direction was presented by Bruna et al. (2013) [20], where they utilized graph spectral theory to define convolution operations on graphs. Since then, many studies have refined and approximated this method to reduce computational costs and enhance applicability [21, 22, 23, 24].

- **Spatial-based methods**: These methods were studied early on but received little attention. Micheli et al. (2009) [25] proposed a non-recurrent model that addressed inter-node dependencies while maintaining the message-passing mechanism from RecGNNs. However, this research did not gain much recognition at the time. Only in recent years have spatial-based ConvGNNs attracted more attention with the emergence of advanced models [26, 27, 28].

2.1.3 Other Developments in GNNs

Besides RecGNNs and ConvGNNs, several other GNN methods have emerged to extend the capability of modeling graph data. Some notable developments include:

- **Graph Autoencoders (GAEs)**: These models use autoencoder architectures to learn compressed representations of graphs, serving tasks such as community detection or link prediction.

- **Spatial-Temporal Graph Neural Networks (STGNNs)**: These models extend GNNs to handle dynamic graph data, where relationships between nodes change over time.

Overall, GNN methods continue to evolve in various directions, leveraging features from RecGNNs, ConvGNNs, and other advanced neural network architectures to tackle a wide range of problems on graph data.

2.1.4 GNNs and Network Embedding

GNNs are closely related to graph embedding or network embedding, a field that has garnered increasing interest from the data mining and machine learning communities [29, 30, 31, 32, 33, 34].

Network embedding aims to represent nodes in a graph as low-dimensional vectors while preserving both the network's structural connections and node attribute information. This allows various graph analysis tasks, such as classification, clustering, and recommendation, to be effectively handled using conventional machine learning algorithms like support vector machines (SVMs).

On the other hand, GNNs are deep learning models designed to process graph-related problems in an end-to-end manner, enabling the extraction of high-level features explicitly.

The key differences between GNNs and network embedding are as follows:

- GNNs are a family of neural network models tailored for diverse tasks.
- Network embedding consists of various methods that all aim to achieve a common objective.

As a result, GNNs can address network embedding tasks through the graph autoencoder framework. Conversely, network embedding also includes non-deep-learning methods such as matrix factorization [35, 36] and random walk-based approaches [37].

2.1.5 GNNs and Graph Kernel Methods

Graph kernel methods were once the dominant techniques for solving graph classification problems [38, 39, 40]. These methods use kernel functions to measure similarity between graph pairs, enabling kernel-based algorithms such as SVMs to perform supervised learning on graphs.

Both GNNs and graph kernel methods map graphs or nodes into a vector space, but their approaches differ:

- Graph kernel methods rely on a fixed (deterministic) mapping function.
- GNNs learn the mapping function from data, allowing for greater adaptability.

Since graph kernel methods require pairwise similarity computation, they suffer from scalability issues. In contrast, GNNs can directly extract features and classify graphs efficiently.

For more details on graph kernel methods, readers may refer to [41].

2.2 Definition

2.2.1 Graph structure

A graph in Fig. 2.1 is formally represented as $G = (V, E)$, where V denotes the set of vertices or nodes, referred to as "nodes" hereafter, while E signifies the set of edges. More precisely, let $v_i \in V$ be indicative of a node, and $e_{ij} = (v_i, v_j) \in E$ be representative of an edge originating from node v_i and terminating at node v_j. In this context, the concept of the node's neighborhood, denoted as $N(v)$, encompasses the set of nodes $u \in V$ in which an edge (v,u) exists within E.

Central to the representation is the adjacency matrix A, a square matrix of dimensions $n \times n$, where n stands for the number of nodes within the graph. Elements A_{ij} of this matrix follows the logic that A_{ij} equals 1 if the corresponding edge e_{ij} belongs to E; conversely, A_{ij} assumes a value of 0 if e_{ij} is not an element of E. Furthermore, the framework accommodates the inclusion of node attributes, denoted as X, where $X \in \mathbb{R}^{n \times d}$. Here, X operates as a node feature matrix, with rows $x_v \in \mathbb{R}^d$ encapsulating the unique feature vector associated with each node v.

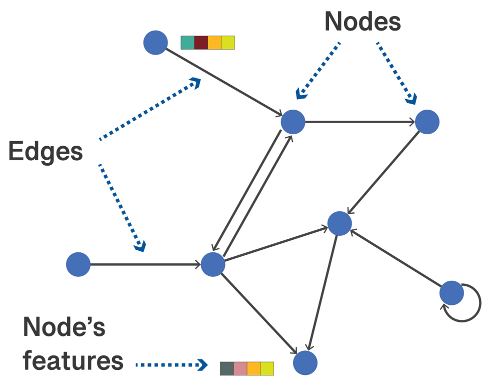


Figure 2.1: Graph structure.

Simultaneously, the graph can incorporate edge attributes represented by X^e. In this context, $X^e \in \mathbb{R}^{m \times c}$ serves as an edge feature matrix, where m represents the number of edges in the graph (the number of rows in the matrix X^e) and c represents the number of features for each edge (the number of columns in the matrix X^e). Each distinct entry, $X^e_{(u,v)} \in \mathbb{R}^c$, housed within X^e characterizes the feature vector corresponding to the respective edge (v,u). This nuanced exposition underscores the capacity of the graph model to holistically accommodate both node and edge attributes, thereby enhancing its suitability for sophisticated real-world applications.

2.2.2 Directed graph

A **directed graph** is a type of graph in which all edges have a specific direction, meaning that each edge goes from one node to another in a determined manner. In other words, if there exists an edge from node u to node v, there is not necessarily an edge from v to u. Directed graphs are commonly used to model asymmetric relationships such as traffic flow, causality relations, or computer networks, where data can only be transmitted in a specific direction.

On the other hand, an **undirected graph** can be considered a special case of a directed graph, where, if two nodes are connected, there exists a pair of edges in opposite directions, ensuring that the relationship between nodes is bidirectional. In an undirected graph, there is no distinction between the starting and ending points of an edge; rather, an edge simply represents a connection between two nodes.

Formally, a graph is classified as **undirected** if and only if its adjacency matrix is symmetric. That is, if there exists an edge between nodes u and v, then there must also be an edge from v to u, ensuring that $A_{uv} = A_{vu}$. This property reflects the symmetry in relationships between nodes and guarantees that the graph can be represented without considering edge directions. Undirected graphs are frequently used to model symmetric relationships, such as friendships in social networks, road connections between cities, or links between elements in a physical system.

2.2.3 Convolutional Graph Neural Networks (ConvGNNs)

Convolutional Graph Neural Networks (ConvGNNs) share a strong connection with Recurrent Graph Neural Networks (RecGNNs). However, instead of iteratively updating node states with contraction constraints as in RecGNNs, ConvGNNs address cyclic dependencies by employing a fixed number of layers, each with distinct weights.

Due to their efficiency in performing graph convolutions and their seamless integration with other neural network models, ConvGNNs have rapidly gained popularity in recent years. These methods can be broadly categorized into two main approaches:

- **Spectral-based Methods**: This approach defines graph convolution using filters derived from graph signal processing techniques. The convolution operation in this context can be interpreted as a process of noise reduction in graph signals, facilitating the extraction of meaningful structural information.

- **Spatial-based Methods**: Building upon the principles of RecGNNs, spatial-based methods define graph convolution by propagating information among neighboring nodes. This facilitates the learning of spatial relationships within the graph structure.

Since the introduction of the GCN model [23], which bridges the gap between spectral-based and spatial-based methods, spatial-based approaches have significantly advanced due to their efficiency, flexibility, and generalization capabilities.

In this study, I focus on the spatial-based approach to effectively leverage information propagation between nodes in the graph.

Spectral-based ConvGNNs

Spectral-based methods have a strong theoretical foundation in graph signal processing [42, 43, 44]. These methods assume that the graph is undirected and is represented by the normalized graph Laplacian matrix, which is defined as:

$$L = I_n - D^{-1/2} A D^{-1/2}$$

where: - D is the diagonal degree matrix, with elements $D_{ii} = \sum_j A_{ij}$, - A is the adjacency matrix representing the connections between nodes, - I_n is the identity matrix of size n.

A key property of the normalized graph Laplacian matrix is that it is symmetric and positive semi-definite, allowing it to be decomposed as follows:

$$L = U \Lambda U^T$$

where: - $U = [u_0, u_1, \ldots, u_{n-1}] \in \mathbb{R}^{n \times n}$ is the matrix of eigenvectors of L, - Λ is the diagonal matrix containing the corresponding eigenvalues, with $\Lambda_{ii} = \lambda_i$.

The eigenvectors of the Laplacian matrix form an orthonormal basis, satisfying the condition:

$$U^T U = I$$

In the context of graph signal processing, a signal can be represented as a vector $x \in \mathbb{R}^n$, where each element x_i corresponds to the signal value at node i. The graph Fourier transform (GFT) is then defined as:

$$F(x) = U^T x$$

Conversely, the inverse graph Fourier transform (IGFT) is given by:

$$F^{-1}(\hat{x}) = U \hat{x}$$

where $\hat{x}$ represents the transformed signal in the spectral domain. Essentially, this transformation projects the input graph signal onto the space spanned by the eigenvectors of the Laplacian matrix. The spectral components $\hat{x}$ serve as coefficients in the linear combination of eigenvectors. Consequently, the original signal can be reconstructed via the inverse Fourier transform:

$$x = \sum_i \hat{x}_i u_i$$

This expression illustrates how a signal can be decomposed into spectral components, providing a powerful approach for processing and analyzing graph signals.

Spatial-based ConvGNNs

Spatial-based methods define graph convolution through message passing between neighboring nodes. Unlike spectral-based approaches, which rely on eigen decomposition of the graph Laplacian, spatial-based methods operate directly in the node domain, making them more scalable and adaptable to various graph structures.

A fundamental operation in spatial-based ConvGNNs is the aggregation of information from neighboring nodes. Given a graph $G = (V, E)$, where V represents the set of nodes and E denotes the edges, the feature representation of a node v at layer $l + 1$ is updated as follows:

$$h_v^{(l+1)} = \sigma\left(W^{(l)} \sum_{u \in \mathcal{N}(v)} \frac{h_u^{(l)}}{|\mathcal{N}(v)|} + b^{(l)}\right)$$

where:

- $h_v^{(l)}$ represents the feature vector of node v at layer l,
- $\mathcal{N}(v)$ denotes the set of neighboring nodes of v,
- $W^{(l)}$ is the weight matrix at layer l,
- $b^{(l)}$ is the bias term,
- σ is a non-linear activation function (e.g., ReLU).

This aggregation process ensures that each node updates its representation by incorporating information from its local neighborhood, thereby capturing structural and relational properties within the graph.

One of the key advantages of spatial-based ConvGNNs is their locality-driven nature. Since these models aggregate information from neighboring nodes in a recursive manner, they are more efficient and flexible for large-scale graphs. Unlike spectral-based methods, spatial-based approaches do not require a fixed graph structure, making them well-suited for dynamic and heterogeneous graphs.

A notable model that bridges the gap between spectral-based and spatial-based approaches is Graph Convolutional Network (GCN) [23], which simplifies the aggregation rule by introducing a normalized propagation mechanism:

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-1/2} \tilde{A} \tilde{D}^{-1/2} H^{(l)} W^{(l)}\right)$$

where:

- $\tilde{A} = A + I$ is the adjacency matrix with self-loops,
- $\tilde{D}$ is the degree matrix of $\tilde{A}$,
- $H^{(l)}$ represents the node features at layer l.

This formulation allows information to be propagated efficiently while maintaining stability during training. Due to its effectiveness, spatial-based ConvGNNs have become the dominant approach in recent years, offering a balance between computational efficiency and expressive power.

In this study, we focus on spatial-based methods, leveraging their ability to dynamically adapt to graph structures and efficiently aggregate node information.

Chapter 3

GCN-Based Solutions for SQL Injection Detection and Sign Language Recognition: Problem and Approach

3.1 Towards Lightweight Based on GCN Model For SQL Injection

3.1.1 Rationale

Although graph convolutional networks (GCN) have been introduced for nearly a decade and have attracted significant attention from the scientific research community, they have not yet been widely applied in many research domains and real-world applications as convolutional neural networks (CNN). One of the main barriers to the widespread adoption of GCN lies in the challenge of input data representation.

Unlike CNN, which operates on Euclidean-structured data such as images or time series, GCN requires input data to be represented as a graph consisting of nodes and edges that define relationships between entities. However, in many real-world scenarios, data is not naturally structured as a graph, and transforming it into this format poses significant challenges, limiting the applicability of GCN beyond academic research.

In this chapter, I explore the potential of GCN by applying it to a classical problem that has not previously been addressed using this method: SQL injection detection. The goal of this study is to develop an optimized GCN model that effectively processes graph-structured input data while improving detection performance.

To achieve this, I propose the model *Towards Lightweight Model Using Non-local-based Graph Convolution Neural Network for SQL Injection Detection*, which integrates non-local-based graph convolution techniques to leverage non-local dependencies within the data.

Through experimental evaluations, this study aims to assess the effectiveness of the proposed approach compared to traditional models, thereby shedding light on the potential of GCN in cybersecurity applications.

SQL injection presents significant risks to web applications and databases, allowing unauthorized access and data breaches. To tackle this problem, we introduce a novel graph-based network, a previously unexplored structure for detecting SQL injection attacks. In this framework, SQL statements are represented as nodes, with edges defined by their interconnections. We propose three graph convolutional neural network (CNN) models: a graph classification approach utilizing a two-layer Graph Convolutional Network (GCN), a graph classification model incorporating a non-local graph convolution derived from a 1x1 convolution (replacing the standard 1x1 convolution), and an enhanced non-local-block module where the 1x1 convolution layers are substituted with GCN layers. These models achieve an accuracy exceeding 99% and inference times below 1 ms across two datasets. Compared to 22 conventional models, our GCN-based approaches offer improved computational efficiency, reduced parameter counts, higher accuracy, and the flexibility to process input sequences of varying lengths, highlighting their potential for real-world cybersecurity applications, particularly in robust SQL injection detection and prevention.

3.1.2 Introduction

With the rapid expansion of the internet, a continuous surge in tools and applications has emerged, enabling seamless access to information, real-time interactions, and task execution from any location. Alongside this progress, however, cybersecurity challenges and threats to users' personal data have intensified, driven by escalating security risks. Among these, SQL injection stands out as a critical issue, drawing significant attention from developers and researchers alike. This widespread attack exploits vulnerabilities in web applications, aiming to illegitimately infiltrate systems—most notably database systems—posing a severe risk to data integrity and security. Such attacks are characterized by their diversity, rapid adaptability, and covert nature, often leading to substantial harm.

According to the OWASP Top 10 list [45], SQL injection ranks third among security vulnerabilities, as illustrated in Fig. 3.1, underscoring the urgent need for develop-

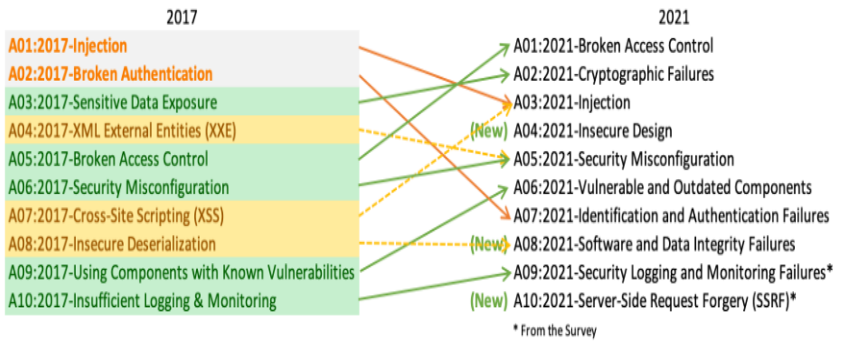

2017
A01:2017-Injection
A02:2017-Broken Authentication
A03:2017-Sensitive Data Exposure
A04:2017-XML External Entities (XXE)
A05:2017-Broken Access Control
A06:2017-Security Misconfiguration
A07:2017-Cross-Site Scripting (XSS)
A08:2017-Insecure Deserialization
A09:2017-Using Components with Known Vulnerabilities
A10:2017-Insufficient Logging & Monitoring
2021
A01:2021-Broken Access Control
A02:2021-Cryptographic Failures
A03:2021-Injection
(New) A04:2021-Insecure Design
A05:2021-Security Misconfiguration
A06:2021-Vulnerable and Outdated Components
A07:2021-Identification and Authentication Failures
(New) A08:2021-Software and Data Integrity Failures
A09:2021-Security Logging and Monitoring Failures*
(New) A10:2021-Server-Side Request Forgery (SSRF)*
* From the Survey

Figure 3.1: Top 10 web application security risks by OWASP [45].

ers to implement robust countermeasures. Alarmingly, 94% of evaluated applications revealed vulnerabilities tied to SQL injection during testing. These weaknesses are categorized into 33 distinct types under the Common Weakness Enumeration (CWE) framework, reflecting their frequent occurrence across various applications.

As a prominent vulnerability in network security, SQL injection triggers profound consequences, jeopardizing the authenticity, integrity, authorization, and overall safety of systems [46]. A notable instance is the 2011 cyberattack on Sony's PlayStation Network, where a skilled hacker group exploited SQL injection flaws. This breach compromised over 77 million accounts, including the theft of around 12 million credit card records [47]. Beyond the exposure of sensitive user data, the attack inflicted economic damages estimated at up to 170 million USD. A comparable incident unfolded in 2017, when attackers exploited SQL vulnerabilities to extract sensitive data from more than 20 universities and government bodies in the UK and US. These cases highlight the critical importance of countering SQL injection techniques to protect information and network infrastructures.

Theoretically, any web application reliant on a database is susceptible to SQL injection risks. As illustrated in Fig. 3.2, inadequate authentication measures or weak protective strategies during the development process can allow attackers to seamlessly inject and execute SQL commands within the system.

This ability grants attackers elevated privileges to query or alter data, leading to significant fallout. First, it creates a hidden risk of exposing critical data, including personal and financial details, as attackers can retrieve information from the database. Second, it enables data modification or deletion, undermining data integrity and causing the loss of essential information. Third, SQL injection can empower attackers to seize control of the system, potentially inflicting direct harm to the system or its components, resulting in considerable time and financial costs for recovery. Finally, such attacks can severely tarnish the reputation of the affected organization or application.

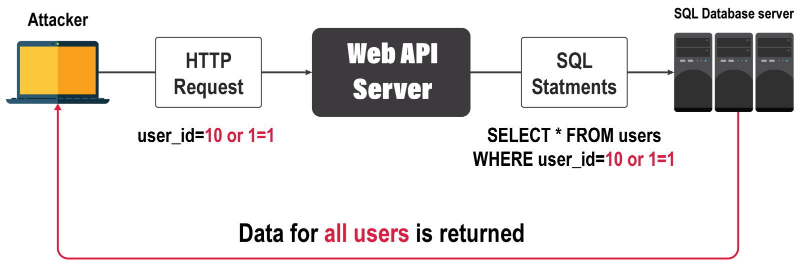


Figure 3.2: SQL injection attack mechanism.

Thus, defending against SQL injection vulnerabilities is vital for ensuring information security and system stability.

A key difficulty, however, stems from the sophisticated stealth of these attacks. Unlike other attack types that produce overt signs, SQL injection can be subtly executed, evading the typical indicators of network intrusions. Attackers often infiltrate through standard web application ports, blending their actions with legitimate client requests, which complicates detection by firewalls. If unnoticed, these attacks can persist undetected for prolonged periods, causing substantial losses.

Most modern web application firewalls employ rule-based or pattern-matching algorithms [48], widely used to shield systems from similar threats. While efficient, these methods struggle to address the full spectrum of SQL injection variants. Machine learning approaches have also been adopted to detect such attacks. Typically, researchers extract distinct features from known attack samples and apply basic machine learning techniques for classification, achieving reasonable accuracy. Yet, the complexity and diversity of SQL injection patterns pose ongoing challenges, requiring continuous model refinement. Although deep learning advancements are increasingly utilized across fields, their application to SQL injection detection often faces overfitting issues. To mitigate this, techniques such as data augmentation, model optimization, and innovative methods have been proposed, yielding promising outcomes.

In this research, we introduce a deep learning model for identifying SQL injection attack code sequences, combining Text2Vec, Graph Convolutional Network (GCN), Long Short-Term Memory (LSTM), attention mechanisms, and non-local network techniques. Our approach begins by converting input text data into vector representations using word embedding to map words into vectors. Next, we construct a graph structure to capture relationships within the sequential text data, linking words based on defined rules. GCN is then applied to address the problem. We compare our model with alternatives, such as those using LSTM and attention, on the same dataset, demonstrating that our proposed model achieves superior accuracy based on experimental results.

This chapter is organized as follows: Section 2 reviews related research efforts in this domain. Section 3 elaborates on the design methodologies and proposed models.

3.1.3 Related Works

SQL injection vulnerabilities have been a persistent and vital focus in web security research for decades, with broader efforts in AI-driven threat detection and news sensors offering complementary insights [49]. Early investigations aimed to highlight the risks and impacts of these vulnerabilities. A significant early contribution by Halfond and Orso [50] outlined the dangers of SQL injection in web applications and described attacker strategies.

As attack methods have advanced, so too have SQL injection techniques. Stuttard's recent study [51] explored sophisticated approaches, such as blind SQL injection, time-based blind SQL injection, and out-of-band SQL injection, providing critical understanding of their variety and reinforcing the need for forward-thinking countermeasures.

Detecting SQL injection is essential for safeguarding security and data integrity. Traditional methods for identifying and preventing these attacks typically rely on rule-based analysis using predefined patterns. Common practices include employing prepared statements or parameterized queries to isolate user inputs from SQL commands. Prepared statements enable developers to predefine query structures and populate them with user data, avoiding direct concatenation. Similarly, parameterized queries ensure a clear divide between user inputs and SQL code, blocking unintended command injections.

However, these conventional approaches require significant manual effort. Researchers often extract features based on prior knowledge and use string-matching techniques for detection, which struggle to adapt to novel attack methods. Their rigidity also limits responsiveness to changing application needs.

Beyond traditional techniques, recent progress in machine learning has introduced innovative ways to apply these methods, including leveraging real-time social media data for cyber-attack prediction, as examined in a Twitter-based survey [52]. Many contemporary studies have used basic machine learning algorithms to detect SQL injection, with feature extraction playing a central role. By analyzing attack command patterns, these efforts enhance model detection capabilities. For instance, Afnan Mahmud Al Badri et al. [53] implemented the AdaBoost algorithm in a Web

Application Firewall (WAF) classifier, achieving notable accuracy. Eman Hosam's research [54] extracted 13 features and tested six machine learning models, reaching 99.6% accuracy. Aidana Zhumabekova [55] evaluated models like naïve Bayes, support vector machines, decision trees, random forests, XGBoost, and CatBoost, all exceeding 99.5% accuracy. Additionally, Kasim Tasdemir [56] used a bag-of-words approach for sentence representation, testing 20 machine learning models. Yet, these methods' success hinges on effective feature extraction, demanding significant manual resources and adaptability to evolving attack diversity.

Over the past decade, deep learning's rapid evolution has entrenched its role across research domains, driven by its powerful feature extraction abilities. This has sparked interest in applying deep learning to SQL injection detection and prevention. Abaimov et al. [57] encoded raw data into patterns and used 1D convolution to demonstrate efficacy. Thalji's team [58] proposed AE-Net, a neural network for text feature extraction, achieving 99% accuracy. Zhang et al. [59] compared Convolutional Neural Networks (CNNs) with traditional machine learning, highlighting CNNs' superior predictive power. Overall, deep learning, particularly CNNs, outperforms conventional methods in SQL injection detection.

Word embedding techniques have recently surged in popularity, finding use in tasks like text classification, knowledge extraction, and question answering. Neural network models based on the distributional hypothesis excel at mapping semantic word relationships into low-dimensional spaces. Though word embedding predates recent advances, hardware improvements and optimization breakthroughs around 2012–2013 elevated neural network models' prominence [60]. The 2013 introduction of word2Vec [61] significantly boosted research interest, paving the way for advanced deep learning applications.

Amid this focus on embedding models, researchers have harnessed them for text feature extraction in deep learning. Xie et al. [62] introduced an elastic-pooling CNN, vectorizing SQL queries with Word2vec and processing them via CNN, surpassing traditional machine learning performance. Gowtham's team [63] used CBOW and SKG for semantic feature extraction, achieving 98% accuracy with machine learning classification. Luo et al. [64] employed the gensim library for embedding, followed by CNN classification, reaching up to 99% accuracy.

Innovative approaches include Zhang's [59] use of a Deep Belief Network (DBN) to monitor HTTP requests for SQL injection detection. Tang [65] applied MPL and LSTM networks to extract features from HTTP sequences, while Qi Li et al. [66] proposed an LSTM-based method for intelligent transportation systems. Further exploration of LSTM variants, such as BILSTM with an attention mechanism [67], has enhanced classification performance.

Recent advancements in natural language processing have produced robust architectures like Transformers [68]. Introduced in 2018, BERT [69] excels in language tasks and could classify SQL queries as character sequences. However, sequence-based methods like LSTM and modern NLP models like BERT, despite their semantic strengths, face high computational costs, a limitation also noted in AI for edge computing in IoT systems [70], reducing efficiency in systems needing rapid, large-scale data processing.

A research gap I've identified is that simple deep learning models using standard convolutions often require fixed input vector sizes, restricting their ability to handle variable-length SQL queries. While larger models can mitigate this, they increase computational complexity, undermining real-time feasibility. Thus, I propose using the flexible, non-Euclidean nature of graphs to create a compact model that processes varying input lengths without fixed-size constraints.

3.1.4 Methodology

To tackle the challenge of SQL injection classification, our approach builds upon foundational techniques. As noted earlier, SQL vulnerabilities primarily stem from developers' oversight in system design, often neglecting to rigorously filter user inputs or implement robust mechanisms to detect suspicious data. Malicious actors exploit these gaps by injecting dynamic SQL code into input fields, enabling them to breach the system and execute harmful commands. Acknowledging this root cause, we recognize that classifying incoming data is a pivotal step. In our view, this data typically manifests as text. Consequently, our research centers on exploring text processing methods—such as word embedding, Recurrent Neural Networks (RNNs), and diverse feature extraction and synthesis techniques—to enhance the identification of hidden risks within data objects through optimized processing and analysis.

Word Embedding with FastText Word embedding is a fundamental method in Natural Language Processing (NLP) that transforms natural language vocabulary into numerical representations. Its primary aim is to map words into vector spaces where semantically similar words are positioned in close proximity.

Introduced in 2017 by Facebook AI Research, FastText [71] is a word embedding technique that leverages the skip-gram model to predict surrounding words for a given word, while employing an n-gram model to generate representations based on subword units. By integrating these approaches, FastText produces detailed vocabulary embeddings, capturing both word-level and sub-word-level information to boost NLP performance.

The probability of a word w given its context context is computed using the softmax function:

$$P(w \mid \text{context}) = \frac{e^{\mathbf{v}_w \cdot \mathbf{v}_{\text{context}}}}{\sum_{\forall w} e^{\mathbf{v}_w \cdot \mathbf{v}_{\text{context}}}} \tag{3.1}$$

where:

- P: The likelihood of word w appearing within a context, defined by its surrounding n-grams.

- $\mathbf{v}_w$ and $\mathbf{v}_{\text{context}}$: Vector representations of the word and its context, derived by combining the vectors of n-grams within the word and its contextual environment.

FastText stands out for its compact and efficient embedding capabilities, particularly in managing out-of-vocabulary words, securing its prominence in word embedding applications. In our study, we aim to uncover the latent SQL language patterns within textual data, using a training dataset composed of text strings. To this end, we adopted the FastText model for word embedding. Its lightweight design enables rapid data processing without compromising research quality, making it an ideal choice. In SQL sentences, the arrangement of keywords and symbols is critical, and FastText excels in capturing these patterns efficiently, avoiding the need for overly intricate or resource-heavy models. A key strength of FastText is its ability to handle words absent from the training data, allowing our model to adapt effectively to real-world scenarios without requiring frequent retraining.

Non-local Network

The idea of attention, first developed in natural language processing [72, 73] to identify global connections between inputs and outputs, is not included here. In recent years, self-attention mechanisms have proven versatile across a range of applications. Additionally, extensions of Convolutional Neural Networks (CNNs) have emerged, incorporating non-local or long-distance dependencies [74] to model spatial relationships among distant data elements.

In architectures employing non-local strategies, self-attention is typically not used to determine attention weights between data points. Instead, these models rely on a non-local interaction function to measure interactions across the input space, facilitating global relationships over varying distances. For an input represented as

$x \in \mathbb{R}^{H\times W\times C}$ and an output as $y \in \mathbb{R}^{H\times W\times C}$, corresponding to the input and output feature maps, the non-local block is formulated as:

$$y_i = \frac{1}{C(x)} \sum_{\forall j} f(x_i, x_j) g(x_j) \tag{3.2}$$

$$y = \text{softmax}(X^T W_\theta^T W_\phi x) W_g x \tag{3.3}$$

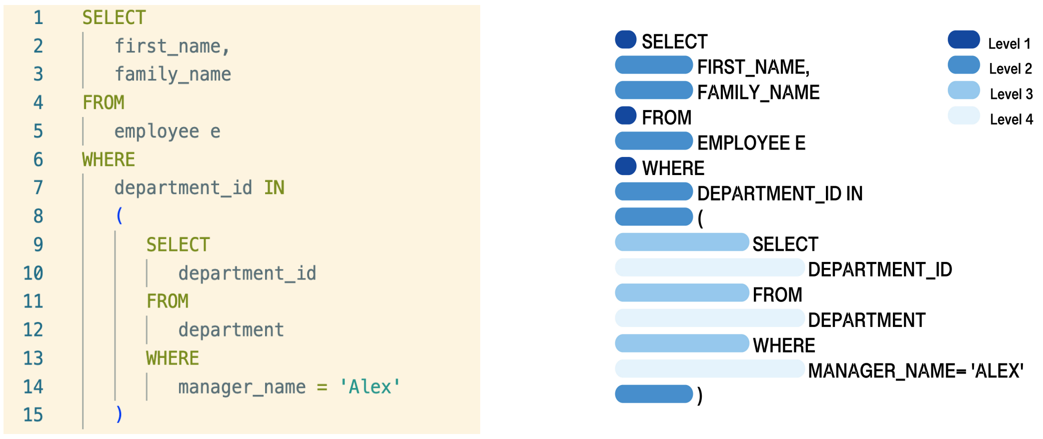
Here, y results from applying the non-local operation across the entire dataset, with W_θ, W_ϕ, and W_g representing weight matrices.

3.1.5 Approach

Proposed Conversion of SQL Queries into Graph Structures

To tackle this challenge, we adopt a graph-based methodology. Our approach focuses on normalizing SQL indentation in the provided input string. For instance, consider the following raw input:

```
SELECT first_name, family_name
FROM employee e
WHERE department_id IN (SELECT department_id
                        FROM department
                        WHERE manager_name='Alex')
```



```
SELECT
    first_name,
    family_name
FROM
    employee e
WHERE
    department_id IN
    (
        SELECT
            department_id
        FROM
            department
        WHERE
            manager_name = 'Alex'
    )
```

(a) (b)

Figure 3.3: SQL commands and normalization into graph form.

To accomplish this objective, we begin by designing an algorithm to normalize SQL statements, reformatting the input data accordingly. Drawing on standard SQL

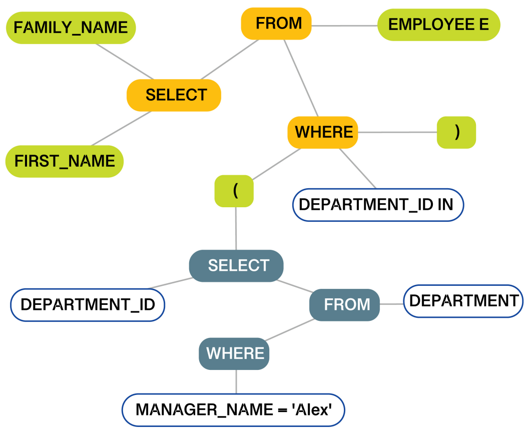


Figure 3.4: SQL graph structure sample.

formatting guidelines [75], we crafted a tailored set of rules to guide the reformatting process. Our aim is to clearly isolate key elements, such as keywords and interrelationships between components, laying the groundwork for generating node structures and their connections in the graph during subsequent stages of the workflow.

1. Organize the SQL statement according to widely accepted SQL indentation standards.
2. Convert all words in the input query to uppercase.
3. Position each keyword on a new line, shifting the remaining content to the following line with an indentation level increased by one.
4. Align all opening and closing parentheses on the same line.
   - Move the content between opening parentheses to the next line, indented by one additional level.
   - Set the indentation of closing parentheses to one level less than the preceding line.

Upon standardizing the SQL statement structure, we produce a multiline format where each line contains one or more words, with critical terms like SQL keywords isolated on individual lines. The hierarchical organization is visually evident through indentation depth, as shown in Fig. 3.3.

To transform this standardized format into a graph, we must precisely define elements such as nodes, edges, node features, and edge features. Within the context of SQL statements, each line in the formatted indentation structure becomes a node in the graph. The indentation depth of each line serves as its hierarchical level index. The connection rules between nodes are outlined as follows:

- Traverse the nodes sequentially from top to bottom.
- Link nodes at higher hierarchical levels to the closest node above them with a lower hierarchical level.
- Connect nodes at the lowest hierarchical level in a top-to-bottom sequence.

These relationships are represented as edges in the graph, which are undirected, as illustrated in Fig. 3.4. With nodes and edges established, a critical next step is to generate features for each node. Using word embedding techniques, the content of each node is mapped to a vector in a multidimensional space, serving as the node's feature set.

Overall Proposed Structure

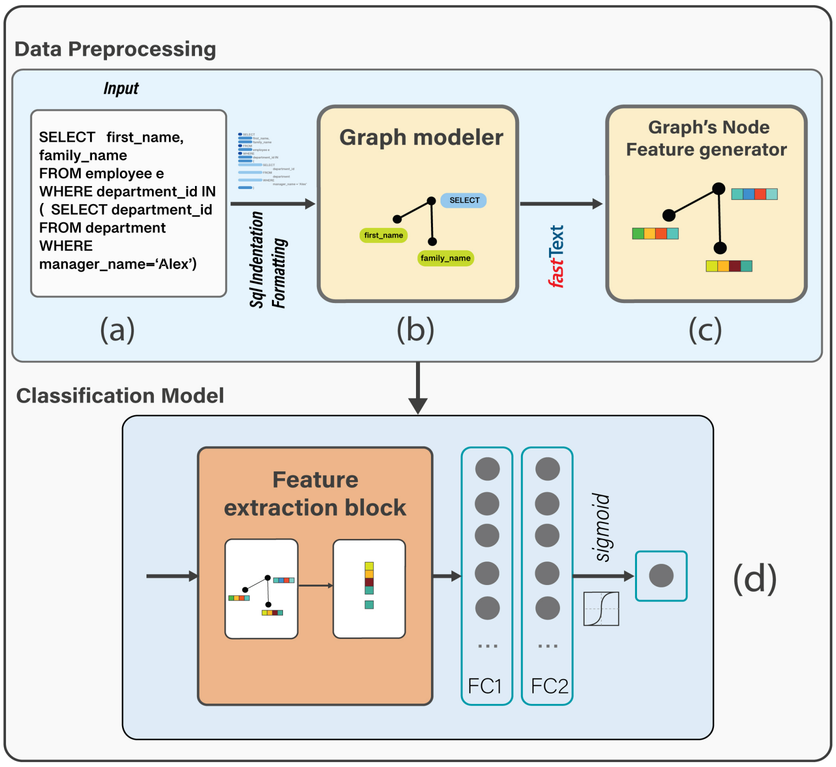

Data Preprocessing
Input
SELECT first_name, family_name FROM employee e WHERE department_id IN (SELECT department_id FROM department WHERE manager_name='Alex')
Sql Indentation Formatting
Graph modeler
SELECT
first_name
family_name
fastText
Graph's Node Feature generator
(a)
(b)
(c)
Classification Model
Feature extraction block
sigmoid
FC1
FC2
(d)

Figure 3.5: Overview of the proposed model structure.

A primary obstacle in detecting SQL injection attacks is the ability to process and differentiate between benign queries and those posing security threats. Our strategy addresses this by converting SQL statements into a graph structure, utilizing structural insights to improve detection accuracy.

The proposed framework, depicted in Fig. 3.5, consists of three main stages. The first stage, the graph modeler, transforms raw data into an indented format and builds the graph structure, as shown in Fig. 3.5. The second stage generates node features using word embedding techniques. The final stage involves a classifier that extracts features from the graph network and performs classification.

In the initial stage, we restructured SQL commands into a more organized format, placing significant keywords—such as SELECT, FROM, WHERE, JOIN—and other syntactic components on separate lines, with indentation levels reflecting their interdependencies. These indentation-based relationships enabled us to connect lines, forming a graph where nodes represent individual lines and edges denote their interconnections based on the SQL statement's content and structure. This graph forms the basis of our graph-based methodology.

In the feature generation stage, depicted in Fig. 3.5, we employed the FastText model to convert node content into 64-dimensional vectors. This process yields an undirected graph complete with nodes, edges, and 64 features per node, encapsulating the SQL statement's internal relationships. This structure supports the application of Graph Neural Networks (GNNs) for information propagation across nodes and edges, enhancing the model's capacity to identify subtle SQL injection traits through graph-based learning and improving classification outcomes.

For the graph classification stage, we utilized graph convolutional layers due to their proficiency in handling graph-structured data. Key benefits of graph convolutions include their adaptability to graph data, ability to process large graphs, and effectiveness with uncertain structures. By leveraging message passing between connected nodes via edges, graph convolution enables the model to capture not only node.

Module 1: Feature Extraction Module with Two Hidden GCN Blocks

We initiate our approach with a straightforward module featuring two hidden Graph Convolutional Network (GCN) blocks, as illustrated in Fig. 3.6. Each block consists of a graph convolutional layer, a normalization layer, and an activation function. The model takes as input a graph with dimensions $n \times 64$, where 64 represents the feature count per node and n varies as the number of nodes in each graph. Throughout the process, the feature dimensionality per node is expanded to 128 and subsequently to 256. Following this expansion, the graph undergoes aggregation via a global mean pooling layer, after which it is passed through two linear layers for classification.

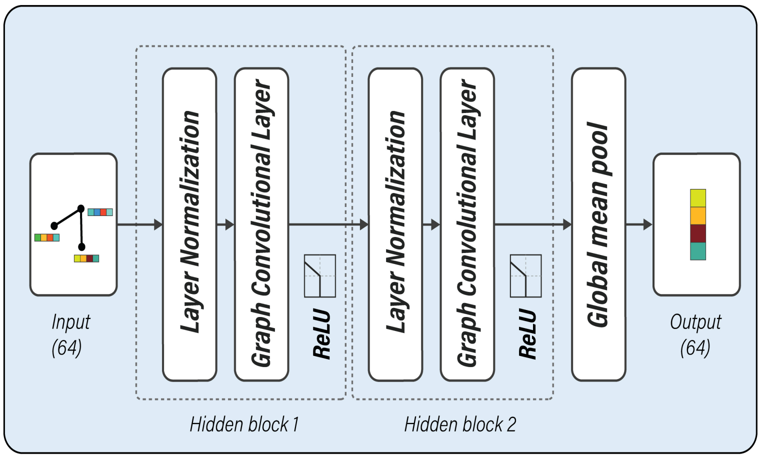


Figure 3.6: Module 1 - Structure of the graph convolutional network with two graph convolutional blocks.

Module 2: Combination of Graph Convolution Blocks and a Non-Local Block

To elevate the model's effectiveness, we investigated the integration of a non-local network, a technique engineered to capture interactions across data regions, both proximate and distant. This approach was adopted with the intent of enabling dynamic feature interactions within the graph, where features are first extracted and refined through graph convolutional layers and global mean pooling to standardize the data shape. Our goal is to enhance the model's classification prowess and overall efficacy for this specific task. Consistent with Module 1, the architecture of Module 2, shown in Fig. 3.7, incorporates two graph convolutional blocks to gather and refine features, increasing the node feature count from 64 to 256 across these blocks. The varying number of nodes in the graph poses both opportunities and hurdles, particularly in stabilizing the data shape. To address this, global mean pooling is applied before the non-local block to ensure a consistent shape, facilitating dynamic feature interactions.

Module 3: Feature Extraction Module with a Modified Non-Local Network Using Graph Convolution Layers

In exploring the architecture of Graph Neural Networks (GNNs) further, we note that GNNs adopt a framework focused on synthesizing local information. During GNN convolutions, data at each node is exchanged with adjacent nodes via a message-passing process. Consequently, a node's information interacts only with nodes within a limited range (k-distance) around it. Typically, each GNN layer aggregates local

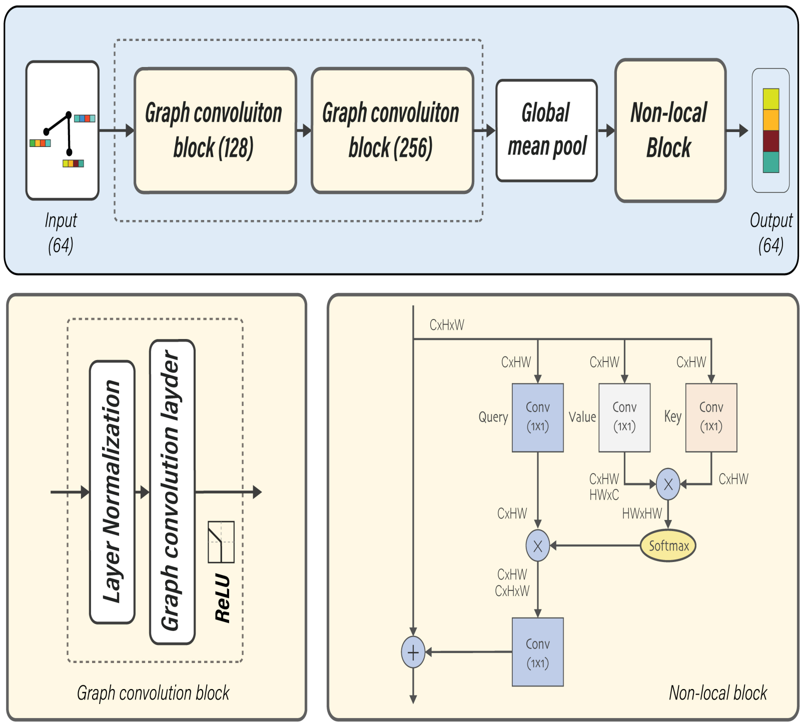


Figure 3.7: Module 2 - Combination model with graph convolutional blocks and a non-local block.

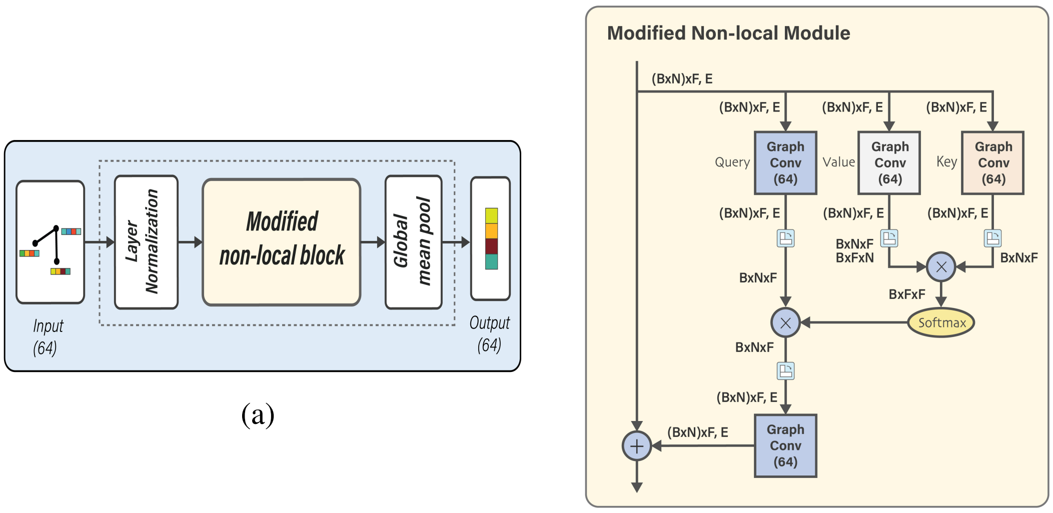


Figure 3.8: Module 3 - A modified non-local module.

data from immediate neighbors, often within a single hop.

While stacking multiple layers could theoretically boost the model's expressive power—enabling distant nodes to communicate—increasing layer depth often introduces noise and fails to optimize information exchange effectively. Furthermore, adding layers can lead to challenges like the over-smoothing problem [76] or the over-squashing issue [77], which may hinder the model's ability to learn and represent intricate relationships within graph data.

Drawing from the traditional non-local network design, we propose a modification, illustrated in Fig. 4.3, by replacing 1x1 convolution layers with GraphConv layers. Originally, 1x1 convolutions apply linear transformations to feature data to reduce dimensionality, enhance adaptability, and lower computational demands while preserving essential information. Substituting these with GraphConv layers maintains the linear transformation functionality but leverages the graph's structure for added benefits. This shift enables local information sharing among nodes before broader interactions occur, reducing computational overhead and allowing the model to exploit graph-specific features. Local elements can thus exchange data prior to engaging with other groups.

A key consideration in adapting the conventional non-local structure with graph convolution is ensuring a fixed node count for seamless reshaping and global interactions. We address this by applying zero-padding, introducing unconnected nodes with a value of 0 to stabilize the graph structure.

3.2 Towards an Efficient GCN Based on MultiHead Attentative for Sign Language Recognition

3.2.1 Rationale

In this study, I focus on enhancing the performance of graph convolutional network (GCN) architectures by incorporating advanced deep learning techniques, refining the structure of convolutional layers, and adjusting the non-local mechanism to optimize the processing of graph-structured data.

The selected experimental task is hand sign language recognition—an important real-world application—while also constructing a dataset for the Vietnamese sign language alphabet. The primary objective of this research is not only to improve the efficiency of GCN models for this specific problem but also to further demonstrate the effectiveness and broad applicability of convolutional models on graph data.

Sign language serves as an essential communication tool for the deaf community, playing a vital role in message transmission and social interaction. Yet, interpreting sign language has presented a persistent challenge for the scientific community. Earlier methods predominantly depended on sensors, but limitations in device practi-

cality and efficiency have restricted their success. Recently, advancements in image processing have opened up a promising new avenue. This research leverages image processing to develop efficient classification models based on graph convolution, aiming to minimize model parameter size and computational demands. We introduce four graph convolution-based techniques, evaluated across three datasets: two publicly available American Sign Language (ASL) datasets and a newly proposed Vietnamese Sign Language (ViSL) dataset. Experimental outcomes reveal exceptional performance, achieving accuracies of 99.53% on the ASL dataset, 99.97% on the MNIST dataset, and 99.80% on the ViSL dataset. These models deliver high efficacy without requiring specialized devices or sensors, outperforming many complex alternatives. The carefully curated ViSL dataset complies with Vietnamese Sign Language standards and matches the performance of the ASL dataset. This work highlights the substantial potential of graph convolution in image recognition tasks, particularly for sign language, fostering improved communication and connection between the deaf community and society.

3.2.2 Introduction

Communication is a cornerstone in the growth and evolution of biological communities. The capacity to share information, ideas, and emotions is fundamental to sustaining and enriching diverse groups, spanning human and animal species alike. Throughout human history, a variety of communication methods have emerged. Beyond spoken and written language, people have employed tools like drawing, sculpting, gestures, and facial expressions to express messages. Among these, listening and speaking are widely regarded as the most intuitive and accessible forms, even for young children. However, for those with hearing impairments, partial or total loss of auditory and speech abilities creates significant barriers to everyday interaction.

Hearing loss, whether affecting one or both ears, can stem from factors such as genetics, aging, noise exposure, infections, ear injuries, or certain medications and toxins. The World Health Organization (WHO) reports that, in 2024, about 5% of the global population—approximately 430 million people, including 34 million children—require hearing rehabilitation [78]. Projections suggest that by 2050, this figure could rise to over 700 million, or 1 in 10 individuals worldwide. People with hearing loss, irrespective of its cause, frequently face communication challenges in modern times.

These language barriers profoundly affect the lives of individuals with disabilities. Sign language stands as their primary mode of communication, bridging gaps caused by hearing and speech impairments. Yet, it remains underused and poorly understood by those with typical hearing, creating a divide between the deaf community and broader society—a challenge yet to be fully addressed. Increasingly, researchers are drawn to developing communication aids for the deaf, seeking solutions to enhance their social engagement.

Sign language, a natural, non-verbal, and visually driven communication system, serves as the primary language for millions of deaf individuals globally. While over 300 distinct sign languages exist worldwide, varying by region and country [79], they share common traits. Sign language comprises two key elements: finger spelling (handshapes) and dynamic gestures (hand movements). Finger spelling uses specific hand and finger positions to represent individual letters, while gestures add expressive depth. This richness and variety in sign language expression present significant hurdles for learning and application.

Researchers have traditionally approached this problem through two main strategies: sensor-based and image-based methods. Sensor-based techniques require users to wear devices like gloves or sensors to detect sign features, whereas image-based methods process camera-captured images without user-worn equipment. In recent years, hand gesture recognition has tilted toward image-based approaches, which offer fewer user restrictions compared to earlier sensor-dependent methods.

Over time, researchers have applied a spectrum of techniques, from basic image processing and comparison to advanced machine learning, including shallow CNNs and deep learning models. These efforts have shown varying degrees of success in mitigating communication challenges for the deaf. In this study, we propose graph convolutional models to classify static signs in sign language, introducing enhancements to improve feature extraction and model performance. We also present a new dataset of static signs for Vietnamese Sign Language, meticulously designed to align with Vietnam's established conventions.

This chapter is structured as follows:

- **Section 2**: Reviews prior research efforts addressing this challenge.
- **Section 3**: Outlines the foundational techniques employed in this work.
- **Section 4**: Details the proposed methodology and its components.
- **Section 5**: Introduces the Vietnamese static gesture dataset, experiments, and evaluates the proposed approach.
- **Final Section**: Offers conclusions and explores future research directions.

3.2.3 Related work

Numerous prior studies have sought to create hand sign recognition systems by integrating information processing with classification techniques. These efforts typically focus on capturing key hand attributes—such as position and shape—before applying machine learning to develop classifiers. As discussed earlier, over the years, researchers have explored hand sign recognition using a variety of devices and methods, broadly divided into two categories: sensor-based and image-based approaches.

Sensor-based studies have garnered significant interest within the research community. For example, Simin Yuan et al. [80] tackled Chinese Sign Language recognition by employing Electromyography (sEMG) sensors on the right arm to gather sEMG signals. These signals were used to train predictive models, including Artificial Neural Networks (ANN) and Support Vector Machines (SVM), achieving a notable accuracy of up to 95.48%. Similarly, Cao Dong [81] utilized affordable depth cameras, such as Microsoft's Kinect, to collect data, applying segmentation and hierarchical model-seeking techniques to pinpoint hand joint locations under kinematic constraints. Their Random Forest model yielded over 90% accuracy for American Sign Language (ASL) recognition. Another approach by Lucas Rioux-Maldague et al. [82] combined depth and RGB images from the Kinect, processed via Deep Belief Networks. Meanwhile, Watcharin Tangsuksant [83] used dual cameras to track six hand points, computing triangle patches from marker triplets for ANN-based classification.

However, these methods require users to rely on specialized equipment, such as sensors or gloves, or fixed systems that lack portability, posing practical challenges and increasing costs. Additionally, many studies emphasize feature extraction from hand sign signals without optimizing classification through Machine Learning (ML) or Artificial Neural Network (ANN) models, resulting in less-than-optimal accuracy.

With advancements in machine learning and deep learning, researchers have turned to processing images from widely available devices like smartphone cameras and webcams. Jungpil Shin [84] estimated the coordinates of 21 hand points, applying SVM and GBM models to classify ASL with 87.60% accuracy. Dewinta Aryanie [85] used a kNN approach, flattening three color channels of input images and testing various k values, achieving a peak accuracy of 99.8% with $k = 3$ on the ASL dataset. To boost precision, complex image processing techniques have also been adopted. CM Jin [86] employed Canny Edge detection and Speeded-Up Robust Features (SURF) for segmentation, followed by classification using K-means clustering, Bag-of-Words, and SVM, reaching 97.13% accuracy. Likewise, Tse-Yu Pan et al. [87] extracted features via Bag-of-Words with SIFT, Hu Moments, and Fourier Descriptors, achieving high accuracy with SVM classification from standard camera inputs.

In recent decades, Convolutional Neural Networks (CNNs) have become a dominant force in image processing and spatial data analysis. Researchers have leveraged CNN-based classifiers for hand sign recognition, yielding impressive results. Many approaches [88, 89, 90, 91] involve CNN architectures with single or multiple hidden layers, processing RGB or grayscale images directly, significantly improving accuracy over traditional ML models. V Jain [92] found that a single-layer CNN outperformed SVM and nearly matched two-layer CNNs. Hybrid models combining CNNs with traditional ML have also been explored. RG Rajan et al. [93] proposed a model with parallel VGG19 and LBP streams, merging feature maps for SVM classification, achieving 98.44% accuracy on the ASL dataset. HBD Nguyen [94] tested combinations like HOG-LBP-SVM, end-to-end CNN, and CNN-SVM, with hybrid models excelling at 98.36% accuracy. Capsule networks, as seen in Md Asif Jalal's work [95], offer another innovative approach to gesture classification.

Regarding diverse input types, studies have utilized RGB and depth images for parallel network models. Research by Bhagat [96] and Aly [97] combined ML and CNNs to extract features from these data types, achieving accuracies of 98.81% and 99.5% for hand gesture tasks.

Recently, graph convolutional methods [3, 98] have gained traction for processing graph-structured data, earning acclaim for their feature extraction efficiency and high performance across applications. However, a key challenge is the need to represent input data as graphs. Converting traditional formats into graph structures requires careful precision to retain essential data traits, ensuring effective feature extraction and prediction. Alongside simpler models, researchers have employed deep learning with transfer learning, leveraging pre-trained models from large datasets to accelerate development. Studies [88, 99, 100, 101] have implemented prominent models like VGG16, VGG19, AlexNet, ConvNeXt, EfficientNet, ResNet-50, VisionTransformer, InceptionV3, and YoloV3. While effective in many domains, not all excel in classification tasks. For instance, YoloV3 achieved only 95% accuracy, lagging behind basic ML models or simpler CNNs, as noted in research reports.

3.2.4 Preliminaries

Attention mechanism

Scaled Dot-Product Attention

Scaled dot-product attention [68] is a mechanism that determines attention weights through a scaled dot-product computation. The process unfolds as follows:

1. **Similarity calculation:** A dot product is performed between the query matrix Q and the key matrix K to generate attention scores.
2. **Scaling adjustment:** These scores are divided by $\sqrt{d_k}$ to mitigate excessively large values when d_k grows, promoting gradient stability.
3. **Weight standardization:** The softmax function normalizes the scaled scores into attention weights, ensuring their sum equals 1 and allowing probabilistic interpretation.
4. **Weighted value aggregation:** The output is computed as a weighted sum of the value matrix V, with the attention weights acting as coefficients.

The attention weight computation is expressed as:

$$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V. \tag{3.4}$$

Multi-Head Attention

Multi-head attention [68] enhances the basic attention framework by enabling the model to simultaneously focus on diverse representation subspaces. The procedure includes:

- First, the queries (Q), keys (K), and values (V) are individually transformed linearly to produce distinct representations for each head. These are then fed into separate scaled dot-product attention layers.
- With h attention heads, h unique outputs are generated. Each head employs its own learned weight matrices W_i^Q, W_i^K, W_i^V, allowing it to emphasize different facets of the input sequence.
- The outputs from all heads are concatenated and processed through a final linear transformation W_o to revert to the original dimensionality.

The computation for each attention head is given by:

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V). \tag{3.5}$$

The complete multi-head attention output is formulated as:

$$\text{MultiHead}(Q,K,V) = \text{concat}(\text{head}_1, \dots, \text{head}_h)W_o. \tag{3.6}$$

Non-Local Network

The concept of self-attention first arose in Natural Language Processing (NLP) [72, 73] to capture complex global relationships between input and output sequences. In recent years, self-attention models have proven versatile across numerous applications, alongside the development of non-local or long-range dependency extensions for Convolutional Neural Networks (CNNs) [102]. The non-local framework is carefully crafted to model interactions not just within local regions but also across distant elements, enhancing the grasp of spatial connections and thereby improving feature extraction and data processing efficiency.

Models employing non-local mechanisms frequently use self-attention to calculate attention weights between data points, enabling global interactions irrespective of their spatial separation in the input domain. Representing the input and output feature maps as $x \in \mathbb{R}^{H\times W\times C}$ and $y \in \mathbb{R}^{H\times W\times C}$, respectively, the non-local block's mathematical expression can be described as follows:

$$y_i = \frac{1}{C(x)} \sum_{\forall j} f(x_i, x_j) g(x_j). \tag{3.7}$$

Alternatively, this can be expressed in matrix form:

$$y = \mathrm{softmax}(X^\top W_\theta^\top W_\phi x) W_g x. \tag{3.8}$$

where:

- y: vector represents the output after applying the non-local expression to the entire dataset.
- W_θ, W_ϕ, W_g: are weight matrices.

3.2.5 Proposed models

To develop a system adept at recognizing hand signs and enabling flexible access to everyday objects, we adopt an image-based approach. Within this framework, we identify two core tasks: extracting and processing features from images and building a classification model.

Originally envisioned as a solution to address classification through graph convolutional techniques, our methodology explores feature extraction, processing, and classification using graph representations. The model's architecture consists of three key elements:

1. **Hand skeleton feature extraction:** This stage concentrates on identifying critical features that depict the hand's skeletal framework.

2. **Graph synthesis feature injection:** This step involves preprocessing to enhance the feature set and create a graph structure incorporating these features.

3. **Classification model:** This component develops models optimized for classifying graph-based representations.

This integrated strategy aims to deliver robust hand gesture recognition, facilitating its practical application in real-world scenarios.

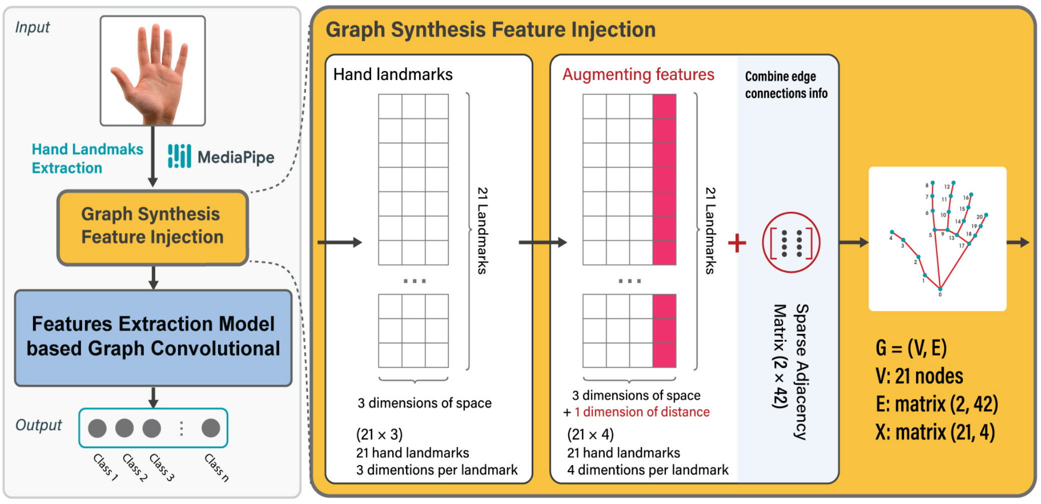


Figure 3.9: Overview of the proposed structure.

Hand Landmarks Extraction

As previously noted, the vocabulary of sign language consists of finger gestures, where each letter is represented by a specific hand and finger configuration. The hand's form is predominantly shaped by its skeletal structure, comprising bones and joints that create a foundational framework. Variations in hand shape, whether due to motion or transformation, manifest as changes in this skeletal system. Thus, a deep comprehension of hand shape is intricately tied to the skeletal framework's structure, and information about hand shape can be used to deduce the skeletal layout, and vice versa.

In conceptual terms, the hand's skeletal structure resembles a graph, with parallels drawn between its bones and joints and a graph's components. Here, joints act as nodes, and bones function as edges linking these nodes within the skeletal graph.

With this foundation, our first task is to extract features of the skeletal system. In a graph context, each node depends on its connections and distinct attributes. Features derived from images include elements like color and depth, but we prioritize the 3D spatial positions of hand joints. Numerous reliable tools exist for extracting or estimating these features from images, and we chose MediaPipe [103] for its compelling advantages.

MediaPipe is a widely used tool in practical applications, valued for its efficiency, reliability, and lightweight design across various devices. It excels in handling 3D data, enabling precise feature generation for key hand point positions, making it ideal for hand image processing. Using MediaPipe, we accurately identify 21 key points on the hand from an image and estimate their 3D coordinates, establishing a strong basis for building graph features of the hand's skeletal structure in later steps.

Graph Synthesis Feature Injection

The effectiveness of a graph model's classification hinges on the quality of its constituent elements, particularly the node features, which in our case stem from the graph synthesis feature injection process (see the workflow in Figure 3.9). To improve these node features, we explored distributing hand feature points evenly around the origin. After collecting these points' attributes, we shifted the origin from a fixed spatial position to one below the index finger. This adjustment was based on the observation that extracted feature points tend to cluster in a specific region relative to the origin. Repositioning the origin nearer the hand's center minimizes changes to this point during shape variations, while other points exhibit more significant shifts, enhancing differentiation among feature indices as the hand transforms. This process yields a feature matrix of hand landmarks with dimensions (21, 3) from the original images and data.

To further enrich the graph's information, we expanded the node features. Similar to other data types, increasing available information lays the groundwork for an effective classification model. By adding distance data for each landmark, we create a feature vector for the hand's node shape, sized (21, 4), where '21' reflects the number of landmarks and '4' indicates the features per landmark. This completes the node feature representation.

For a full graph representation, we integrate node and edge information into a unified network. Edge connectivity determines whether the graph is directed or undirected; we opt for an undirected graph, allowing bidirectional access between nodes. To support information flow in graph convolutional models, each node connects to itself. Following MediaPipe's connectivity sequence [104], we construct a sparse adjacency matrix of size (2, 42). We do not define inter-edge relationships, leaving edge features unspecified and treating all edges equally.

Combining the landmark feature matrix (21, 4) with the sparse adjacency matrix (2, 42) produces a complete graph representation capturing the hand's spatial properties. This graph includes 21 nodes, where V denotes the set of 21 vertices representing the nodes, and E represents the edges. The sparse adjacency matrix (2, 42) outlines node connectivity, while the feature matrix X, with dimensions (21, 4), encapsulates each node's attributes, with each row corresponding to a node.

Proposed classification models

Type-1 model: Basic 2-layer graph convolutional model

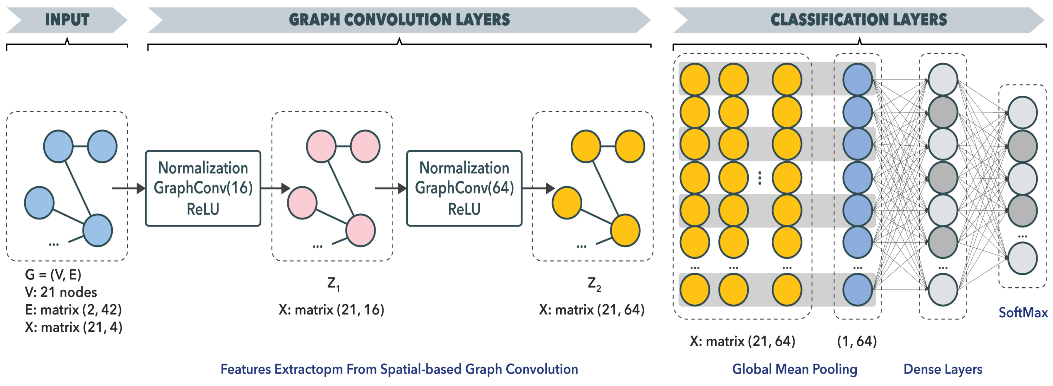


Figure 3.10: Type-1 model structure.

In this research effort, aimed at creating a system based on graph classification techniques, we begin by establishing a basic classification module (Figure 3.10) featuring two hidden blocks of a graph convolutional network. Each block in this structure includes a normalization layer, a graph convolutional layer (GraphConv [105]), and an activation function. As defined in the feature extraction and preprocessing stages, the input consists of an undirected graph with 21 nodes linked by an adjacency matrix E of size (2, 42), where each node possesses four unique features. Through graph convolution, the feature space expands progressively, reaching 16 features in the first hidden block and 64 features in the second. After feature extraction and synthesis, the graph is aggregated using a global mean pooling layer before passing through two dense layers for classification.

Type-2 Model: Enhanced 2-Layer Graph Convolutional Model

Starting with preprocessed input data comprising nodes—each defined by four distinct features: 3D spatial coordinates and distance from the origin—we noted that these attributes reflect diverse aspects and may influence hand shape recognition dif-

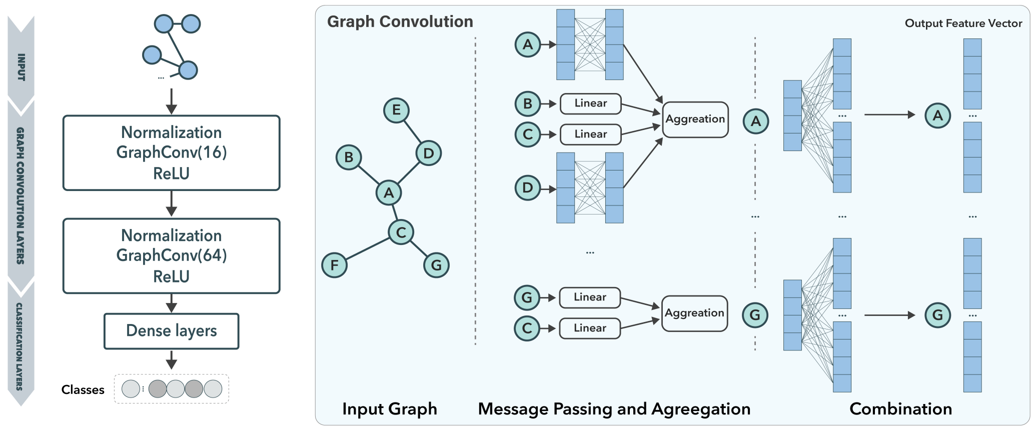


Figure 3.11: Type-2 model structure: Implementation of graph convolution enhancements on the Type-1 model.

ferently. To improve the model's performance, we conducted a deeper analysis of the convolution mechanism within the GraphConv [105] layer and introduced targeted improvements.

The core convolution process in GraphConv involves two key phases: message passing and information integration. In message passing, data from adjacent vertices is transmitted to the target vertex, followed by aggregation using functions like "addition," "mean," or "max pooling." The aggregated information from neighboring vertices is then combined via a linear transformation to produce the final vertex representations.

Recognizing the dispersed and varied nature of the input layer features, we added an extra linear layer before the message passing stage. This step enables a transformation of the feature space, allowing the model to generate new representations by integrating diverse, distinct features across multiple dimensions. This enhancement strengthens the model's ability to capture complex features and relationships within the spatial and distance data, ultimately improving its overall performance.

$$f^{(t)}(v) = \sigma\left(v' \cdot W_1^t + \sum_{w \in N(v)} w' \cdot W_2^t\right). \tag{3.9}$$

where:

$$v' = f^{(t-1)}(v) \cdot \mathbf{W}_{\mathbf{b}}^{\mathbf{t}}$$

$$w' = f^{(t-1)}(w) \cdot \mathbf{W}_{\mathbf{b}}^{\mathbf{t}}$$

Here:

- σ: Represents an activation function.

- W_1^t, W_2^t, W_b^t: Weight matrices for the respective transformations.
- $N(v)$: The set of neighbors of vertex v.
- $f^{(t-1)}$: The function from the previous time step.

The enhanced model (Figure 3.11) maintains the framework of the Type-1 model but replaces the standard GraphConv layer with a revised version, incorporating the improved convolution mechanism.

Type-3 Model: Enhanced 2-Layer Graph Convolutional Model with Skip-Connection and Multi-Head Attention

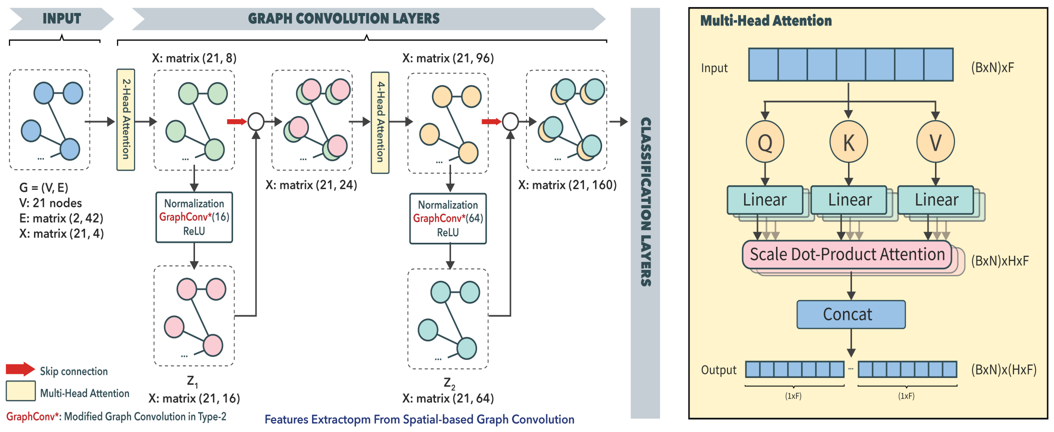


Figure 3.12: Structure of the Type-3 model.

To improve model performance, we introduce an innovative architecture (Figure 3.12) that integrates skip connections and multi-head attention mechanisms. Our primary goal is to enhance the convolutional layer's ability to extract information by emphasizing key node features and retaining maximum data across network layers.

To prioritize node interactions based on their importance, we incorporate an attention-based module. This component calculates weights derived from feature relationships, enabling the identification and assessment of essential features within the graph's nodes. This approach produces a refined graph representation, reducing noise and highlighting critical data elements.

Following this focused representation from the attention mechanism, we process the data using a graph convolution module. Combining attention with graph convolution enhances information handling across the graph. The graph convolution module efficiently extracts and disseminates information throughout the network, utilizing the sharpened representation from attention to create stronger feature representations for each node.

This module retains the adapted graph convolutional layer from Model 2 and includes three hidden blocks (Block 1 and Block 2), each featuring a graph convolutional layer preceded by an attention layer. The first attention layer, before the initial hidden block, uses two heads due to the limited diversity and complexity of node features at this stage, allowing focus on significant inter-node relationships. After the first hidden block expands features to 16 dimensions, the second attention layer employs four heads, enabling the model to address more complex feature interactions and improve information extraction efficiency.

Moreover, skip connections are added to streamline the information flow. These connections directly transfer data from the attention layer's output, merging it with the graph convolutional layer's results. This ensures effective integration of the attention-focused data with the convolutional output, maximizing the use of information from both processes to yield the final representation.

The Proposed Model: Modified Non-Local Network with Graph Convolutional Layer

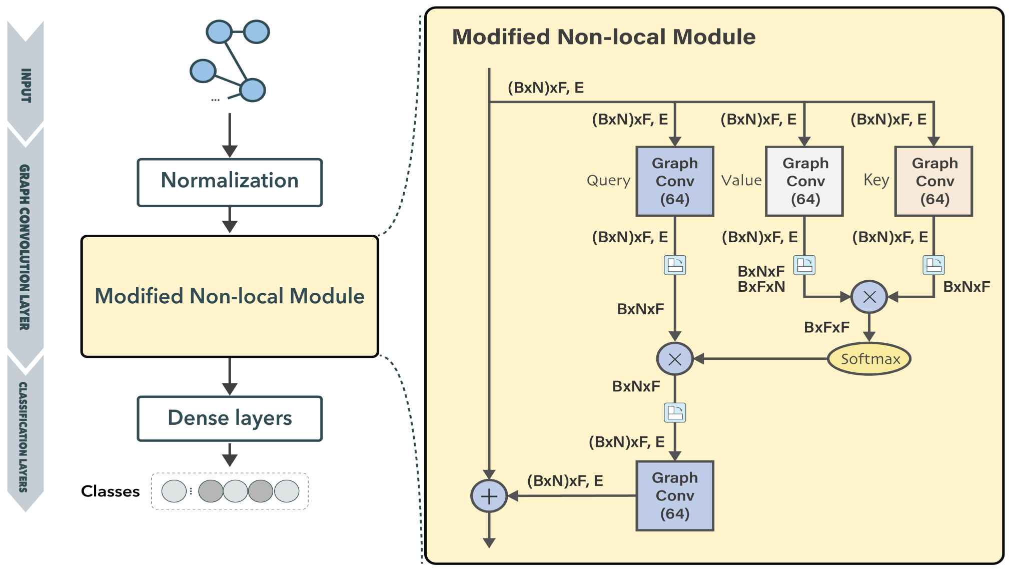


Figure 3.13: Proposed model featuring a modified non-local block structure.

Upon analyzing the convolutional mechanism in GraphConv, we identified its reliance on edge connections for information propagation and aggregation as a limitation. While effective for local information sharing among neighboring nodes, it restricts interactions to immediate connections, hindering the model's capacity to produce novel feature representations. Theoretically, stacking multiple GraphConv layers could address this, but practical implementation often leads to information homogenization and noise. Research also indicates that excessive layering can cause over-smoothing [76] and over-squashing [77] issues.

To overcome these limitations, we propose a tailored non-local block model designed for direct graph data processing (Figure 3.13). Although graph sizes vary with node counts, tasks like processing 21 hand joint points allow us to standardize data size. We replace the original 1x1 convolutional layers with GraphConv layers, extending this change across the model. Initially, 1x1 convolutions performed linear transformations to reduce dimensionality, enhance flexibility, and lower computational load while preserving key data. GraphConv layers, while also performing linear transformations, leverage the graph structure, enabling local information exchange before broader interactions. To ensure node interactions remain graph-specific, we carefully manage data resizing.

Given the input node features' low dimensionality (only 4 features), we increase the internal feature dimensions to 128 within the block, then reduce them to 64 before applying global mean pooling and classification.

Chapter 4

Experiments, Result and Discussion

4.1 Towards Lightweight Model Using Non-local-based Graph Convolution Neural Network for SQL Injection Detection

4.1.1 Datasets and Experiments

This study aims to develop two distinct model types: one prioritizing accuracy and another optimized for inference speed.

The motivation for these models is twofold: First, we emphasize accuracy due to its critical importance in identifying SQL injection attacks. Second, we target reduced inference time, as real-world server systems must efficiently process a high volume of requests within tight time constraints, ensuring responsiveness and compatibility with practical hardware environments.

The experimental setup includes the following core elements:

- **Accuracy evaluation on dataset-I:** We assess the models' precision in detecting SQL injection attacks through extensive training and testing on dataset-I.
- **Inference time analysis on both datasets:** Recognizing the need for rapid processing, we thoroughly measure the models' inference times on dataset-I, selected to evaluate efficiency under conditions requiring quick responses to numerous requests.

Dataset

To evaluate our proposed models' performance in terms of accuracy and inference speed, we employ two datasets. HTTP servers typically handle requests from diverse client applications, such as Android devices, web browsers, and desktop software, involving operations like authentication and data retrieval from users and potential attackers. For real-time malicious code detection, we implemented a two-layer security firewall at the server's application layer. The initial layer, situated between middleware and the router, leverages a dataset of collected malicious payloads [106, 107].

Sourced from Kaggle, the SQL datasets comprise raw data on SQL injection attacks and benign traffic from various websites. After thorough cleaning, both datasets include a label column classifying SQL queries, with 0 representing non-malicious queries and 1 indicating malicious ones.

Dataset-I [106], termed SQL injection dataset I, is used to gauge the accuracy-focused model's effectiveness. It contains 30,873 entries, with 38% being malicious SQL statements and 62% benign, gathered from SQL injection attacks and normal traffic across websites, refined for detection purposes.

Dataset-II [107], known as SQL injection dataset 2, assesses the lightweight inference time model's improvements. A polished version of the Kaggle dataset, it includes 19,537 benign entries (Label = 0) and 11,382 malicious entries (Label = 1), over 1.6 times the benign count. Both datasets were divided into an 8:1:1 ratio for training, validation, and testing, respectively.

Experimental Environment

All experiments in this study are conducted on the Google Colab server, harnessing its robust computational resources for training and evaluating the proposed models.

The CPU specifications of the server are detailed as follows:

- Processor: Intel(R) Xeon(R) CPU @ 2.20GHz
- CPU Family: 6
- Model: 79
- Cache Size: 56,320 KB
- Cores: 1
- Siblings: 2

The experiments are tailored to address the complexities of deep learning-based SQL injection detection, with the server's setup supporting efficient parallel processing and memory management, essential for training and assessing sophisticated models.

To simulate real-world conditions, GPU usage is disabled during testing, reflecting scenarios where servers must process requests without GPU support. This ensures a thorough evaluation of model performance across diverse operational settings.

Metrics Performance

After training our proposed model on the training set, we evaluated its performance on the test set using several metrics to measure its effectiveness in detecting SQL injection. These metrics include accuracy during training and testing, precision, recall, F1-score, and parameter count. The mathematical definitions of these metrics are provided below.

The accuracy metric measures the proportion of correctly classified samples and is expressed as:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FN + FP} \tag{4.1}$$

Precision, a vital indicator of the probability that a sample is correctly classified, is defined as:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{4.2}$$

Recall, also known as sensitivity or the true-positive rate, reflects the fraction of positive samples accurately identified. It is calculated as:

$$\text{Recall} = \frac{TP}{TP + FN} \tag{4.3}$$

The F1-score, which balances precision and recall to provide a holistic evaluation of model performance. is given by:

$$\text{F1-Score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{4.4}$$

In these formulas, TN denotes the true-negative rate, representing the number of normal requests correctly identified. TP indicates the true-positive rate, corresponding to the number of malicious requests accurately detected. FN represents the false-negative rate, reflecting the number of normal requests misclassified, while FP denotes the false-positive rate, indicating the number of malicious requests incorrectly predicted.

4.1.2 Evaluation of the proposed accuracy model

In this experimental analysis, we benchmark three proposed models—(1), (2), and (3)—against a range of models, including an LSTM model, BERT variants, and various machine learning models, to assess their effectiveness. The models evaluated are detailed as follows:

1. Proposed Model 1 (P1): Utilizes the original 2-GraphConv-layer module.
2. Proposed Model 2 (P2): Employs a 2-GraphConv-layer module augmented with non-local blocks.
3. Proposed Model 3 (P3): Features a modified non-local block, substituting 1x1 convolutional layers with graph convolutional layers.
4. LSTM Model (M4).
5. BERT Models: bert_uncased_L-2_H-128_A-2 (M5) and bert_uncased_L-4_H-256_A-4 (M6).
6. Machine Learning Models (18 models): Encompasses classifiers such as AdaBoost (M8), Bagging (M9), Decision Tree (M10), Extra Trees (M11), K-Neighbors (M12), Linear Support Vector Classification (M13), Logistic Regression (M14), Multi-Layer Perceptron (M15), Multinomial Naive Bayes (M16), Nearest Centroid (M17), Nu-Support Vector Classification (M18), One-vs-One (M19), One-vs-the-Rest (M20), Passive Aggressive (M21), Linear Perceptron (M22), Ridge (M23), SGD (M24), XGB (M25), and CNN (M7). All models were retrained and tested on dataset-I to analyze correlation and efficiency.

For the experimental process, the three proposed graph convolutional models transformed input data into graphs, with node features derived by embedding words into 64-dimensional vectors. Similarly, other models underwent preprocessing to convert input data into 64-dimensional word embeddings. Machine learning (ML) and Convolutional Neural Network (CNN) approaches were applied. The NLTK library provided the English stop-word list, while sklearn's CountVectorizer converted text into vector representations based on word frequencies.

To evaluate the proposed models' performance, experiments were conducted using five metrics: training accuracy, testing accuracy, precision, recall, and F1-score, defined as follows:

- **Training Accuracy**: Represents the proportion of correctly classified data points as SQL attacks relative to the total number of data points in the training set, including both attacked and healthy SQL statements. High training accuracy reflects effective learning from the training data.
- **Testing Accuracy**: Indicates the proportion of data points correctly identified as SQL attacks compared to the total data points in the test set, encompassing

both attacked and healthy SQL statements. High testing accuracy demonstrates strong generalization to unseen SQL injection data beyond mere training set performance.

- **Precision**: For predictions of positive cases (detecting SQL attacks), precision is the ratio of true positives (correctly identified SQL injections) to all predicted positives (including normal statements misclassified as attacks). High precision signifies minimal errors in labeling normal SQL statements as malicious.
- **Recall**: For actual positive cases (SQL attacks), recall is the ratio of true positives (correctly identified SQL injections) to all actual positives (including attacks misclassified as normal). High recall indicates effective detection of SQL injections, reducing missed cases.
- **F1-Score**: The harmonic mean of precision and recall, providing a balanced measure of model performance. A high F1-score reflects an optimal trade-off between precision and recall, minimizing both false positives (normal statements misclassified as attacks) and false negatives (attacks misclassified as normal) at low rates.

In Table 4.1, the three proposed models—P1, P2, and P3—demonstrate exceptional performance across all evaluated metrics, achieving training and test accuracies above 99.9%. These results outperform traditional models and architectures, such as CNN (M7) and various machine learning (ML) approaches, and slightly exceed more advanced frameworks like LSTM (M4) and BERT (M5, M6). This highlights their remarkable capabilities in learning, generalization, and accurate prediction on unseen data. Among the proposed models, P2, which incorporates a non-local block following a 2-layer graph convolution, stands out with a precision of 100%, indicating no instances where normal statements are misclassified as malicious. The LSTM model (M6) also reaches peak precision, reflecting its strength in managing extended sequences by capturing information from distant contexts. For recall, the proposed models P1, P2, and P3 record impressive values of 99.92%, 99.83%, and 99.92%, respectively. The nearest centroid classifier (M17) similarly excels in this metric, nearing perfection and suggesting minimal false negatives for normal statements alongside a high detection rate for SQL attacks. However, balancing precision and recall is essential. The proposed models, alongside contemporary architectures like LSTM and BERT, achieve this equilibrium, with F1-scores surpassing 99%.

Among the three proposed models, Model P1, a straightforward graph convolutional model with two layers, delivers exceptional results, exceeding 99.9% across

Table 4.1: Experiments in Dataset-I for Proposed Accuracy Models with the Dimension of Feature 64, Compared to the Other Models of LSTM, BERT, Transformer, and ML.

Model	Train Acc	Test Acc	Precision	Recall	F1-Score	Param
Proposed 1 (P1)	99.94	99.97	100	99.92	99.96	99,201
Proposed 2 (P2)	99.98	99.93	100	99.83	99.92	99,209
Proposed 3 (P3)	99.92	99.94	99.92	99.92	99.92	29,153
Model 4 (M4)	99.70	99.51	100	98.73	99.35	107,905
Model 5 (M5)	99.88	99.80	99.81	99.66	99.73	11,171,074
Model 6 (M6)	99.87	99.88	99.89	99.78	99.84	172,929
Model 7 (M7)	98.22	96.04	98.96	90.70	94.65	174,273
Model 8 (M8)	93.04	93.43	84.39	97.66	90.54	-
Model 9 (M9)	97.93	92.20	89.32	90.37	89.85	-
Model 10 (M10)	98.33	92.00	89.85	89.47	89.66	-
Model 11 (M11)	98.33	93.26	90.49	91.94	91.21	-
Model 12 (M12)	87.32	84.52	88.27	75.70	81.50	-
Model 13 (M13)	98.10	94.28	88.69	96.22	92.30	-
Model 14 (M14)	96.34	93.39	85.41	97.12	90.89	-
Model 15 (M15)	98.32	95.79	90.80	98.17	94.34	-
Model 16 (M16)	96.88	95.06	89.96	97.04	93.36	-
Model 17 (M17)	78.34	77.30	41.23	100.0	58.38	-
Model 18 (M18)	85.40	83.95	61.31	95.55	74.69	-
Model 19 (M19)	98.10	94.28	88.69	96.22	92.30	-
Model 20 (M20)	98.10	94.28	88.69	96.22	92.30	-
Model 21 (M21)	98.00	94.86	90.06	96.38	93.11	-
Model 22 (M22)	97.74	94.00	91.54	92.82	92.18	-
Model 23 (M23)	96.68	93.63	86.15	97.02	91.27	-
Model 24 (M24)	96.78	93.63	86.15	97.02	91.27	-
Model 25 (M25)	93.94	93.67	86.15	97.14	91.32	-

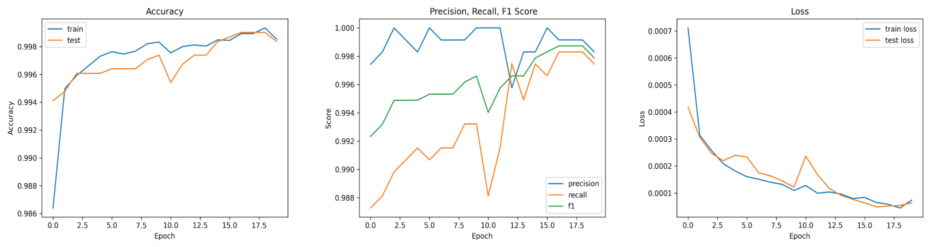
Table 4.2: Experiments in Dataset-I for Proposed Lightweight Models with the Dimension of Feature 16, Compared to the Other Models of LSTM, BERT, and CNN.

Models	TrainAcc	TestAcc	Precision	Recall	F1	IT(ms)	Params
Proposed LW1	99.84	99.71	99.73	99.47	99.60	0.641906	2,209
Proposed LW2	99.90	99.80	99.64	99.82	99.74	0.878548	2,217
Proposed LW3	99.66	99.60	99.91	99.02	99.46	1.363550	5,345
Model 4 (M4)	99.65	99.54	99.73	99.03	99.38	1.390529	21,057
Model 5 (M5)	99.88	99.80	99.81	99.66	99.73	12.261137	4,386,178
Model 6 (M6)	99.87	99.88	99.89	99.78	99.84	26.610154	11,171,074
Model 7 (M7)	98.22	96.04	98.96	90.70	94.65	83.955687	174,273

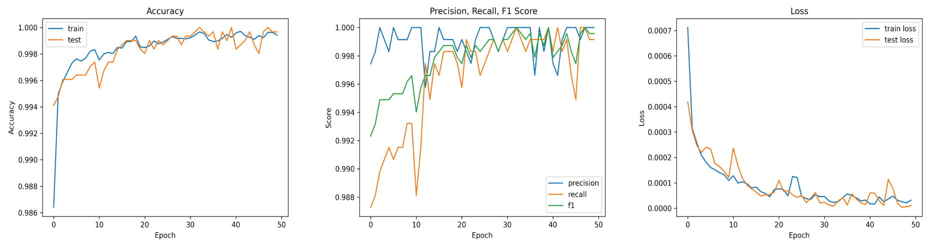
all metrics with a modest parameter count of 99,201. This closely mirrors the performance of Proposed Model 2 (P2), which integrates a 2-layer graph model with a non-local network, achieving similarly high metrics, including a slightly superior training accuracy of 99.98%. Both models maintain low parameter counts, highlight-

Table 4.3: Experiments in Dataset-II for Proposed Lightweight Models with the Dimension of Feature 16, Compared to the Other Models of LSTM, BERT, and CNN.

Models	**TrainAcc**	**TestAcc**	**Precision**	**Recall**	**F1**	**IT(ms)**	**Params**
Proposed LW1	99.84	99.90	99.91	99.83	99.87	0.60049	2,209
Proposed LW2	99.85	99.87	99.91	99.73	99.82	0.863814	2,217
Proposed LW3	99.56	99.42	99.73	98.69	99.21	1.378740	5,345
Model 4 (M4)	99.58	99.70	100	99.22	99.61	1.329851	21,057
Model 5 (M5)	99.78	99.83	99.82	99.74	99.78	7.449558	4,386,178
Model 6 (M6)	99.72	99.87	99.91	99.74	99.82	26.236166	11,171,074
Model 7 (M7)	98.06	97.01	98.64	93.34	95.92	88.75645	174,273



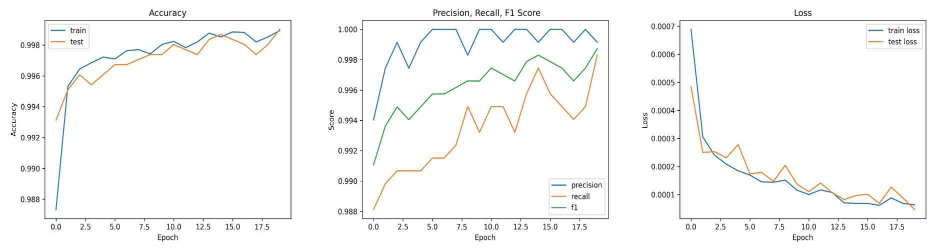
(a)



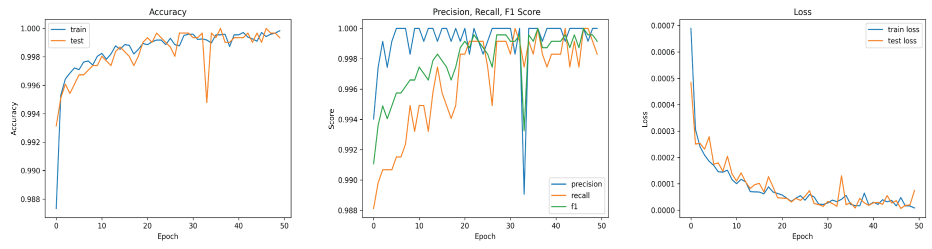
(b)

ing the efficacy of combining graph structures with non-local approaches for high accuracy. Proposed Model 3 (P3), a modified non-local model replacing 1x1 convolutional layers with graph convolutional layers, yields comparable metrics to P1 and P2. However, its parameter count drops significantly to 25,000—just a quarter of P1 and P2's—making it far lighter than other models.

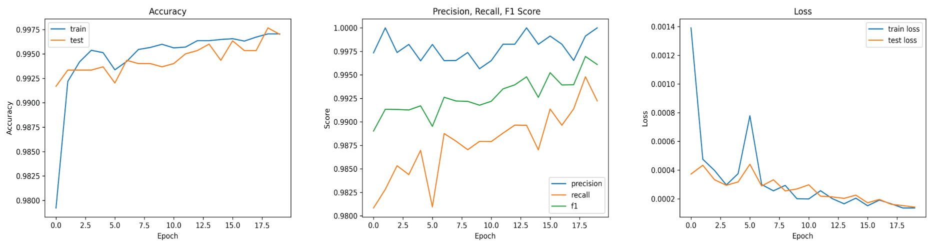
As depicted in Fig. 4.1, evaluation metric curves illustrate variations across epochs. Examining epochs 20 and 50 for models P1, P2, and P3 reveals that test set performance closely mirrors training set results, underscoring the strong learning and generalization abilities of these graph-based models. Overall, the Graph Convolutional Neural Network (GCNN) approach proves more effective than both traditional models (e.g., CNN and ML) and modern architectures (e.g., LSTM and BERT) for this task, excelling in the quest for optimal accuracy.



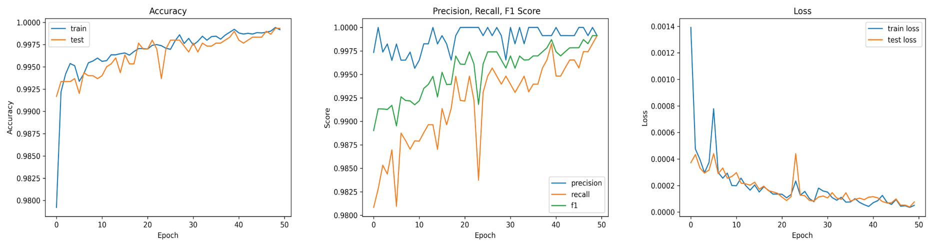
(c)



(d)



(e)



(f)

Figure 4.1: Illustrative graphs associated with accuracy, precision, recall, F1 score, and loss for the three proposed models of P1, P2 and P3. (a) P1 at epoch 20. (b) P1 at epoch 50. (c) P2 at epoch 20. (d) P2 at epoch 50. (e) P3 at epoch 20. (f) P3 at epoch 50.

4.1.3 Evaluation of the Proposed Lightweight Model

In the subsequent experimental phase, we refined the high-accuracy models P1, P2, and P3 into lightweight variants—LW1, LW2, and LW3, respectively—tailored for real-world hardware deployment. Our lightweight model criteria prioritize minimizing inference time, crucial for real-time processing, and reducing parameter counts while preserving accuracy. We achieved this by optimizing parameters in hidden layers, input data, and fully connected layers. Specifically, reducing input embeddings from 64-dimensional to 16-dimensional vectors, limiting feature expansion in hidden layers, and downsizing fully connected layers resulted in models with far fewer parameters yet sustained high accuracy. This optimization proved successful, with models LW1 and LW2 achieving training precision, test precision, precision, recall, and F1-scores above 99.4% on Dataset-I and over 99.8% on Dataset-II, as detailed in Table 4.2 and Table 4.3.

The key goal of parameter reduction is to enhance processing speed, vital for server systems managing numerous requests in tight timeframes. We tested inference time by processing 2,000 data points per model, recording the duration, and averaging the results to evaluate the performance-speed relationship academically.

Consequently, inference speeds were impressively swift, with LW1 and LW2 recording 0.6 ms and 0.8 ms on both datasets—less than half the time of LSTM (M4), 30 times faster than BERT (M5, M6), and roughly 100 times quicker than various machine learning models. Additionally, their parameter counts hover around 2,200, markedly lower than comparable models, reinforcing the precision and efficiency of graph-based approaches with reduced processing times.

However, Model LW3 exhibits a minor performance drop when feature dimensions shrink from 64 to 16. Although it retains high scores on both datasets, all metrics decline, and its parameter count rises 2.5 times compared to its baseline, contrasting with a 0.25 factor at 64 dimensions.

For Dataset-II, comparisons with reference [108], shown in Table 4.4 and Table 4.5 at a 16-dimensional feature size, reveal that LW1 and LW2 outperform in test accuracy, recall, and F1-score, with precision only 0.05 lower. Meanwhile, LW3 underperforms relative to the others at this dimension.

Table 4.4: Comparison of Experimental Results in Dataset-II Between Our Proposed Models and the Model in [108].

Models	**Test Accuracy**	**Precision**	**Recall**	**F1**
LW1	99.90	99.91	99.83	99.87
LW2	99.87	99.91	99.73	99.82
LW3	99.42	99.73	98.69	99.21
[108]	99.86	99.96	99.66	99.81

In summary, the proposed models achieve high scores in precision, recall, F1, and accuracy, coupled with a very small number of parameters. However, upon closer evaluation, at the 64-dimensional feature level, model (P3) operates efficiently with

Table 4.5: Comparison of Differential Results Between Proposed Models and the Model in [108].

Models	Test Accuracy	Precision	Recall	F1
LW1	0.04	-0.05	0.17	0.06
LW2	0.01	-0.05	0.07	0.01
LW3	-0.45	-0.18	-1.04	-0.61

excellent results, especially in terms of significantly fewer parameters than the models of P1 and P2. Nevertheless, after reducing the dimension to 16 to seek optimization for speed, the number of parameters becomes unfavorable and even increases for this model. On the other hand, proposed models of P1 and P2 consistently demonstrate effectiveness across both datasets and different feature dimensions, reinforcing their efficiency.

4.1.4 Analysis of model adaptability and computational feasibility

Model adaptability

SQL injection attacks are highly unpredictable, exhibiting significant diversity and complexity that cannot be anticipated in advance. Moreover, attack patterns and exploitation techniques continuously evolve over time, posing substantial challenges to defensive mechanisms. Consequently, SQLi detection and prevention methods must ensure adaptability to emerging attack variants while optimizing maintenance costs.

With this objective in mind, our study focuses on evaluating the flexibility and generalization capability of machine learning models when trained on various datasets. Specifically, we train three proposed models using embeddings of 16 and 64 dimensions from a given dataset and subsequently assess their performance on independent datasets to examine their adaptability to unseen data.

Table 4.6 is a list of six datasets, including the two previously used datasets and four additional ones.

Table 4.6: Number of samples in each database

	DS 1 [109]	DS 2 [110]	DS 3 [111]	DS 4 [112]	DS 5 [113]	DS 6 [114]
Sample count	30873	30905	148326	107439	98078	98271

We trained our models on dataset 3 [111] due to its large size, with the expectation that the models would generalize well and perform effectively on other datasets. The data was split into three subsets: training, validation, and testing, in an 8:1:1 ratio.

Experimental results (Table 4.7) demonstrate that machine learning models trained on dataset 3 exhibit remarkable flexibility and adaptability when tested on independent datasets, including dataset 3 [111], dataset 4 [112], dataset 5 [113], dataset 6 [114], dataset 1 [109], and dataset 2 [110]. With 16-dimensional embeddings, all three models (LW1-16, LW2-16, LW3-16) achieved an average accuracy exceeding 99%. Notably, LW1-16 recorded the highest test accuracy (99.59% on dataset 1) and F1-scores ranging from 99.15% (LW2-16 on dataset 4) to 99.57% (LW3-16 on dataset 3). These results indicate that the models not only perform robustly on new data but also maintain high performance despite variations among independent datasets.

Increasing the embedding dimensionality to 64 led to a substantial performance improvement, particularly in adapting to new datasets. The LW1-64 model achieved an average accuracy of 99.8% across all six test sets, with an F1-score reaching 99.85% on dataset 2. Meanwhile, LW2-64 recorded an impressive test accuracy of 99.91% on dataset 1 and an F1-score of 99.88% on dataset 2. Even LW3-64, which showed slight variations on dataset 6 (F1-score 99.77%), maintained an overall outstanding performance, with an average test accuracy of 99.8% and an F1-score consistently above 99.66% across all datasets. This improvement not only reflects the ability to learn richer representations from high-dimensional embeddings but also confirms superior generalization, as the models effectively handle unseen features in independent datasets.

A key highlight is that the models, particularly those trained with 64-dimensional embeddings, did not experience significant performance degradation when transitioning from the training dataset (dataset 3) to new test sets, demonstrating their ability to adapt to different data distributions. For instance, on dataset 5—an independent dataset—LW2-64 achieved an F1-score of 99.81% and a test accuracy of 99.82%, while LW1-64 performed even better, with an F1-score of 0.9974 and a test accuracy of 99.75%. Similarly, on dataset 2, all three 64-dimensional models maintained F1-scores above 99.85%, proving their capability to handle diverse real-world scenarios. These results confirm that training on a large and diverse dataset has enabled the models to achieve high flexibility, making them well-equipped to counter emerging SQLi attack patterns in unseen datasets.

Model computational feasibility

To evaluate the complexity and real-world deployability of the LW1, LW2, and LW3 models, we consider two key factors: FLOPs (floating-point operations, reflecting computational cost) and the number of parameters (related to model size and memory requirements). Table 4.8 indicates that at both 16- and 64-dimensional

Table 4.7: Test accuracy and F1-score comparison for models Lw1, Lw2, and Lw3 with 16 and 64 embeddings (test accuracy | F1-score)

data	16-dimensional embedding			64-dimensional embedding		
	LW1	LW2	LW3	LW1	LW2	LW3
DS 3	99.36 | 99.38	99.53 | 99.55	99.55 | 99.57	99.82 | 99.82	99.86 | 99.86	99.81 | 99.82
DS 4	99.22 | 99.25	99.10 | 99.15	99.14 | 99.19	99.55 | 99.57	99.65 | 99.67	99.64 | 99.66
DS 5	99.37 | 99.34	99.47 | 99.45	99.45 | 99.42	99.75 | 99.74	99.82 | 99.81	99.76 | 99.75
DS 6	99.26 | 9935	99.45 | 99.52	99.49 | 99.55	99.78 | 99.81	99.81 | 99.83	99.74 | 99.77
DS 1	99.59 | 99.44	99.61 | 99.47	99.57 | 99.41	99.89 | 99.85	99.92 | 99.89	99.89 | 99.85
DS 2	99.56 | 99.40	99.59 | 99.44	99.54 | 99.38	99.89 | 99.85	99.91 | 99.88	9989 | 99.85

Table 4.8: Computational complexity of LW models (FLOPs | Parameters)

Model	16-dimensional embedding (FLOPs | Parameters)	64-dimensional embedding (FLOPs | Parameters)
LW1	2896.0 | 2209	39040.0 | 99201
LW2	5920.0 | 2217	41088.0 | 99209
LW3	16736.0 | 5345	66848.0 | 142401

embeddings, the models maintain relatively low complexity, ensuring fast processing speed and effective responsiveness in real-world scenarios involving high request volumes.

At the 16-dimensional embedding level, LW1 records only 2,896 FLOPs and 2,209 parameters, making it the most lightweight model in terms of both computational cost and size. This suggests that LW1 can execute extremely quickly, making it ideal for systems requiring real-time responses. LW2 exhibits a slight increase to 5,920 FLOPs and 2,217 parameters, yet remains within a low complexity range, ensuring that processing speed is not significantly impacted despite the added complexity. Meanwhile, LW3, with 16,736 FLOPs and 5,345 parameters, is relatively heavier but still lightweight enough to be deployed on resource-constrained devices, with processing times expected to remain suitable for practical applications.

At the 64-dimensional level, model complexity increases to support enhanced learning capabilities while maintaining computational efficiency. LW1-64 reaches 39,040 FLOPs with 99,201 parameters, LW2-64 registers 41,088 FLOPs and 99,209 parameters, while LW3-64 records 66,848 FLOPs and 142,401 parameters. Although these values are significantly higher than those at the 16-dimensional level, the FLOPs remain moderate compared to more complex deep learning models, which often require millions of FLOPs. This indicates that even in the 64-dimensional configuration, the models remain sufficiently fast for real-time batch request processing, particularly when deployed in distributed systems or on optimized hardware.

A notable observation is that the increase from 16 to 64 dimensions does not lead to an exponential rise in complexity but rather follows a reasonable linear progression, balancing performance (as demonstrated in evaluation results) and processing speed. With FLOPs remaining below 70,000 (in LW3-64) and parameter counts optimized below 150,000, these models are well-suited for real-world deployment, particularly in SQLi detection applications that demand high-speed processing at scale. Furthermore, their compact parameter size reduces memory requirements, facilitating seamless integration into production systems without necessitating expensive specialized hardware. In summary, all three models, at both 16- and 64-dimensional levels, exhibit outstanding potential in terms of speed and practicality, making them well-equipped to handle millions of requests in real-world scenarios without overloading system resources.

4.1.5 Conclusion for Towards Lightweight Model Using Non-local-based Graph Convolution Neural Network for SQL Injection Detection

Throughout this work, we have thoroughly examined and compared 25 different models, spanning from traditional machine learning models to the latest transformer-based models, LSTM, all in the context of SQL injection detection, specifically the proposed graph models. Our analyses revealed that while transformer models, LSTM, and BERT may achieve higher accuracies, they often come with the drawback of significantly higher computational demands. In contrast, classical machine learning models, especially demonstrated remarkable efficiency, balancing high detection accuracy with lower inference time. In the task of SQL injection detection, we successfully constructed a graph network, a novel approach in this context. Through the principles of node creation and node connection, utilizing fastText for words embedding into vectors, each SQL statement is defined as a graph. We proposed two deep learning models based on graph convolution and variation of graph convolutional layer in non-local newtork. These models demonstrated extremely high accuracy, exceeding 99%, with a very small model size. The inference time of the models was significantly reduced, only half or one-fifth of traditional models, and even one-tenth of NLP processing models, with the proposed model (P1) being a graph classification model with 2 GCN layers and model (P2) being a graph classification model using non-local graph convolutional layers. These models were tested on datasets I and II, comparing them with 22 different models. The results showcased improved accuracy, considerably fewer model parameters, and astonishingly fast inference speeds of less than 1ms for the proposed graph models of P1 and P2 in SQL injection detection. Both models demonstrated consistency across different datasets and feature dimensions, affirming their effectiveness. As for proposed model P3 operated effectively and delivered excellent results, particularly with significantly fewer parameters compared to the two preceding proposed models. However, after reducing the dimension to 16 to optimize for speed, the number of parameters became unfavorable and even increased for this model. In conclusion, our study not only provides a comprehensive comparison of SQL injection detection models but also introduces a novel, efficient approach. This approach has potential applications in real-world cybersecurity systems. We believe that our contributions lay the groundwork for further research in this area, stimulating the development of even more effective and efficient systems for detecting and mitigating SQL injection attacks.

4.2 A New Efficient Optimized Graph Convolutional Neural Network based Multi-Head Attentative for Sign Language Recognition

4.2.1 Datasets and Experiment

New dataset: Vietnamese sign language dataset (ViSL)

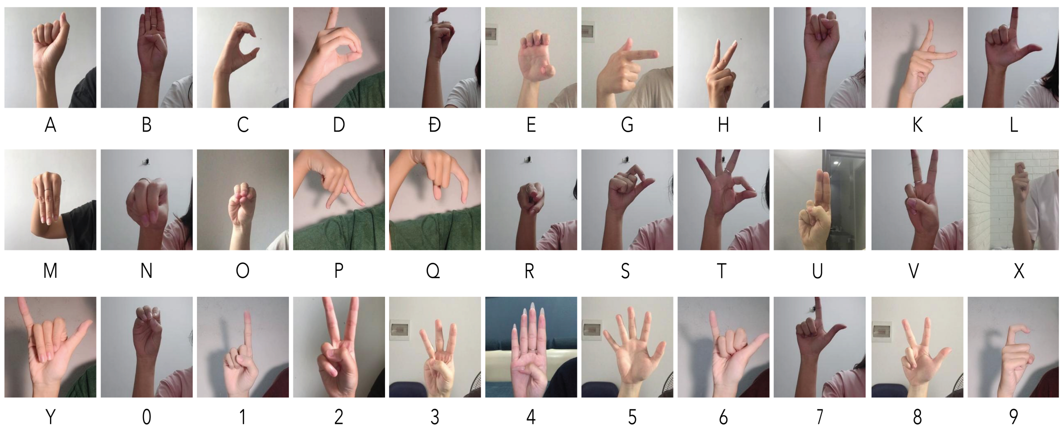


Figure 4.2: Vietnamese sign language (ViSL) dataset.

This study presents a groundbreaking effort in sign language research: the creation of a detailed Vietnamese Sign Language (ViSL) dataset. While most existing datasets focus on American Sign Language (ASL) or English alphabetic symbols, there is a notable absence of datasets designed for the unique linguistic and cultural features of

Table 4.9: Vietnamese sign language (ViSL) dataset properties

Property	**Detail**
Dataset images and labels	• Total images: 118,800 • Labels: 33 • 23 static letters (A-Y) • 10 numbers (0-9) • Images of each label: 3600
Format	JPG
Size of image	1080x1920
Color schema	RGB

Vietnamese sign language. To address this gap, our work seeks to develop a carefully constructed dataset that captures the depth and variety of Vietnamese sign language.

The data collection process was executed with precision to ensure both diversity and fidelity. The steps involved are outlined as follows:

1. **Data acquisition:** We gathered 12 datasets in total, split evenly into 6 datasets for the Vietnamese alphabet and 6 for numbers. Consistency in sign execution and positioning was maintained throughout the collection process. Participants were organized into balanced groups, each tasked with recording videos for 33 characters, with each character captured in two 30-second videos.

2. **Video capture process:** Each video required subtle hand movements to vary the hand's orientation, ensuring comprehensive coverage. After recording, participants performed cross-checks to verify that the quality and progress met established standards. Involving multiple individuals enriched the dataset with diverse perspectives and angles, enhancing its representativeness.

3. **Classification and annotation:** Post-collection, the data was sorted and labeled to enable accurate model training and improve recognition performance. Precise classification and labeling of each frame were critical to this step.

Robust data collection is essential for effective hand gesture recognition models. To ensure quality, we enforced specific requirements: subtle hand rotations during capture to expand gesture variety and improve model generalization; well-lit settings with minimal background clutter to reduce noise and focus on gestures; standardized use of the left hand with the palm facing the camera to prevent misinterpretation;

and uniform practices among collectors to maintain consistency and dataset integrity. Adhering to these rigorous guidelines fosters a high-quality dataset capable of training reliable gesture recognition models, yielding enhanced accuracy and performance.

After individual data collection, group cross-checking was conducted to confirm the accuracy and completeness of each video before advancing to frame extraction. Extracted frames for each character were organized into a folder structure to streamline feature extraction using MediaPipe, supporting subsequent steps. During extraction, frames were resized to a maximum width of 1920px while preserving their aspect ratios.

Our dataset stands apart from existing sign language datasets in several ways. It includes Vietnamese-specific characters like "đ," absent from the English alphabet, and excludes non-Vietnamese characters for a focused representation. These distinctions make it a vital resource for building robust Vietnamese sign language recognition models. A key strength is its reusability as the first comprehensive Vietnamese sign language dataset published, offering potential for expansion and application in areas like deep learning and machine learning. However, it currently lacks dynamic characters, a limitation tied to the model's focus on static frames rather than sequences forming full signs. Future work will aim to incorporate dynamic gestures to advance the model's capabilities.

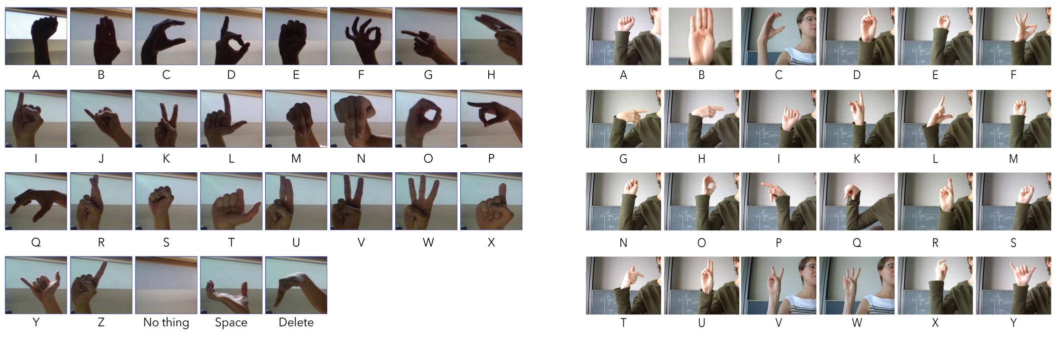
The resulting images meet strict quality criteria, showcasing clear hand gestures free from artifacts like streaks, blur, or noise.

The dataset consists of 118,800 meticulously curated images representing 33 unique labels: 23 static letters from A to Y, 10 digits from 0 to 9, and the Vietnamese letter "đ." This inclusion enhances the dataset's utility and highlights its ability to reflect Vietnamese sign language nuances. Data was collected from seven team members—male and female—ensuring a diverse and representative sample. Captured images meet high standards, free of distortions, and vary in angle, position, and hand size (considering finger and palm dimensions). Stored in ".jpg" or ".jpeg" formats, image sizes range from 720 x 1280 to 1920 x 3413 pixels, using the RGB color model (see samples in Figure 4.2).

Additionally, the dataset features 33 static Vietnamese characters, differing from English-based datasets by excluding "w," "f," "j," and "z" while adding "đ" to reflect Vietnamese traits. Looking ahead, we plan to include letters like "ă," "â," "ê," and "ô," represented through behavioral sequences, a feature unique among sign language datasets. This dataset marks a pivotal advancement in sign language recognition technology, particularly for Vietnam. Its availability paves the way for researchers and developers to devise innovative deep learning and machine learning solutions tailored to

Vietnamese sign language. Furthermore, it establishes a benchmark for future dataset development, underscoring the value of cultural inclusivity in technological progress.

Evaluation datasets: ASL and MNIST datasets



(a) (b)

Figure 4.3: Experimental datasets. (a): ASL dataset, (b): MNIST dataset

In this research effort, we expanded our evaluation by including two additional hand sign datasets alongside our proposed Vietnamese dataset. These datasets were employed to test the effectiveness of our proposed models, offering valuable insights into their performance while deepening our understanding of hand sign traits, ultimately enhancing the robustness of our models.

The first dataset [109] is a comprehensive collection of 87,000 images grouped into 29 classes, representing the 29 letters of American Sign Language (ASL) from A to Z (Figure 4.3a). Each image is sized at 200x200 pixels, providing a substantial and varied resource for assessing model performance. Beyond detailed hand sign data, it includes three unique classes—SPACE, DELETE, and NOTHING—crucial for practical applications and enriching our grasp of sign classification and processing.

The second dataset [110], known as Sign Language MNIST (Figure 4.3b), is a publicly available resource from Kaggle, offering a fresh perspective on American Sign Language analysis. It features 24 static alphabet classes (omitting J and Z due to their dynamic nature), with a total of 27,455 images at 28x28 pixels. Modeled after the classic MNIST dataset, it replaces handwritten digits with hand sign images. Utilizing this dataset allowed us to benchmark model performance across diverse classification tasks, improving our comprehension of the complexity and variety of ASL hand signs.

Experimental Environment

All experiments in this study were performed on Google Colab servers, leveraging their robust computational capabilities for training and evaluating our models. The CPU specifications of the server are detailed below:

- Processor: Intel(R) Xeon(R) CPU @ 2.20GHz
- CPU Family: 6
- Model: 79
- Cache Size: 56,320 KB
- Cores: 1
- Siblings: 2

The experiments were carefully crafted to meet the sophisticated requirements of deep learning-based image classification. The server's setup supported efficient multitasking and memory management, essential for successfully training and testing intricate deep learning models.

Metrics Performance

After training our proposed model on the training dataset, we evaluated it on a separate test set, calculating several metrics to gauge its ability to recognize hand signs. These metrics include accuracy during training and testing, precision, recall, F1-score, and parameter count.

Accuracy measures the proportion of correctly classified samples, determined as the ratio of true positives (TP) plus true negatives (TN) to the total sample count:

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FN+FP}. \tag{4.5}$$

Precision reflects the fraction of true positive samples correctly identified out of all samples classified as positive:

$$\text{Precision} = \frac{TP}{TP+FP}. \tag{4.6}$$

Recall indicates the share of true positive samples correctly classified relative to all actual positive samples:

$$\text{Recall} = \frac{TP}{TP+FN}. \tag{4.7}$$

The F1-score provides a balanced evaluation of precision and recall, calculated as their harmonic mean to assess overall model performance:

$$\text{F1 Score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}. \tag{4.8}$$

In these formulas, TP (True Positive) signifies the count of correctly identified positive samples, TN (True Negative) denotes correctly identified negative samples, FN (False Negative) represents positive samples misclassified, and FP (False Positive) indicates negative samples misclassified.

4.2.2 Evaluation of the proposed model accuracy

Table 4.10: Experimental results on two datasets for four models

	ASL dataset					MNIST dataset					
Model	Train Acc	Test Acc	Pre ci- sion	Re call	F1	Train Acc	Test Acc	Pre ci- sion	Re call	F1	Param eters
Type-1	98.36	98.72	98.73	98.72	98.72	83.77	87.78	88.45	88.78	87.79	6928
Type-2	98.29	98.73	98.75	98.73	98.73	91.42	94.44	94.82	94.44	94.41	8308
Type-3	99.66	99.16	99.17	99.16	99.16	99.46	99.17	99.22	99.17	99.17	41248
Proposed	99.98	99.53	99.54	99.53	99.53	99.92	99.97	99.97	99.97	99.97	26208

In this research endeavor, we undertook a holistic evaluation of the efficacy of four distinct classification models across two heterogeneous datasets: one expansive (ASL dataset) and one diminutive (MNIST dataset). These models, denoted as Type-1, Type-2, Type-3, and the Proposed-Model, were scrutinized with the aim of discerning the optimal model for classification tasks.

Initiating with the Type-1 model, it marks a pivotal stride in utilizing a rudimentary framework comprising two fundamental GraphConv layers. While demonstrating commendable efficacy on the ASL dataset, boasting Train accuracy, Test accuracy, Precision, Recall, and F1-score metrics all surpassing the 98% threshold, its performance wanes significantly when confronted with the notably diminished MNIST dataset, registering a mere 87.78% accuracy on the test subset. This emphasizes a negative impact resulting from limited data availability, thereby diminishing the model's effectiveness in comparison to the MNIST dataset. Despite its modest parameter count, tallying a mere 6928 parameters, this attribute may be construed as an advantage for applications constrained by computational resources.

In a bid to augment model performance via convolutional logic refinements within the GraphConv layer, the Type-2 model has garnered superior outcomes compared

Table 4.11: Comparison of experimental results in ASL dataset between our proposed model and other studies

Method	Related study	Acc
SVM	[84]	87.60%
Light GBM	[84]	86.12%
EfficientNet	[115]	94.30%
AlexNet	[116]	94.47%
ResNet-50	[116]	98.88%
AlexNet	[99]	99.39%
GoogLeNet	[99]	95.52%
ConvNeXt	[100]	99.51%
AlexNet	[100]	99.50%
Vision transformer	[100]	88.59%
EfficientNet	[100]	99.95%
ResNet-50	[100]	99.98%
Modified non-local network with GCN	Our proposed	99.53%

Table 4.12: Comparison of experimental results in MNIST dataset between our proposed model and other studies

Method	Related study	Acc
HOG+SVM (linear kernel)	[117]	90.71%
Random Forest	[118]	65.57%
SVM (linear kernel)	[118]	79.83%
MLP (2 hidden layers)	[118]	75.68%
Shallow CNN	[119]	95.26%
The single layer CNN	[92]	97.34%
The double layer CNN	[92]	98.58%
LeNet	[120]	82.19%
CapsNet	[120]	88.93%
Capsnet augmented	[120]	95.08%
Deep CNN	[92]	97.62%
Modified non-local network with GCN	Our proposed	99.97%

to its Type-1 counterpart across both datasets. Across the ASL and MNIST datasets, this iteration has achieved performance benchmarks of 98.73% and 94.44% respectively, with a marginal uptick in parameter count to 8308. This enhancement can be ascribed to the model's adaptability and heightened generalization prowess when effectuating GraphConv layer transformations. However, the model continues to reflect the repercussions of data constraints, albeit exhibiting improvement over the Type-1 model.

The Type-3 model, iteratively honed from the Type-2 variant by integrating methodologies such as inter-layer skip connections and attention mechanisms for fine-grained feature node processing, has markedly surpassed its antecedents, notably on the MNIST dataset. Despite a substantial escalation in parameter count to 41248, this model has demonstrated consistent efficacy, with both datasets breaching the 99% threshold, namely, 99.16% for the ASL dataset and 99.17% for the MNIST dataset. This underscores the salient impact of input data diversity within each hidden layer on overarching performance.

Lastly, the Proposed model has eclipsed its predecessors on both datasets, consistently attaining above 99% on the training subset and 99.53% for the test subset of the ASL dataset, along with a staggering 99.97% for the MNIST test subset. Despite featuring a mere 26208 parameters, approximately 60% fewer than the Type-3 model, its precision substantially outshines on both training datasets, even under data constraints. This phenomenon may mirror the efficacy of leveraging a non-local network conjoined with graph data amalgamation, facilitating adept exploitation of global spatial information and inter-data point relationships.

Our experimental results have demonstrated that within similar studies utilizing the same dataset, our proposed models outperform many others, particularly machine learning models. Leading models such as Resnet50 and EfficientNet in other practical applications also achieve accuracy levels not significantly divergent from our proposals. Notably, their high performance not only manifests in classification prowess but also in their compactness, utilizing fewer than 50,000 parameters. These proposed models not only enable high performance but also minimize resource requirements, rendering our models flexible and easily deployable across various devices and platforms. This attests to the effectiveness and soundness of the approach presented.

Figure 4.4 depicts the evolution of evaluation metrics across epochs for the four models on ASL dataset and MNIST dataset. Through these curves, we gain an overview of each model's performance. Clearly, the proposed model consistently outperforms others, as metrics such as Train accuracy, test accuracy, Precision, Recall, and F1 remain consistently high with significant deviations from other models. Notably, this model tends to converge faster, exhibiting steep gradients in early stages and early stability in subsequent stages. Additionally, we observe good alignment between evaluation results on the test and training sets, indicating efficient learning and generalization capabilities based on the proposed method. Similar analyses can be conducted when considering the performance of each epoch on dataset 2.

Experiment results on this dataset reveal significant performance differences between the two proposed models and the other two models, as well as significantly faster convergence rates. From these analyses, it can be concluded that the application of proposed methods not only yields good performance in experimental models but also in similar studies, playing a crucial role in identifying the most suitable model for the problem at hand.

4.2.3 Valuation proposed model performance on Vietnamese sign language (ViSL) dataset

Building upon previous studies with publicly available datasets, we conducted a series of experiments on a self-constructed dataset of Vietnamese sign language. The

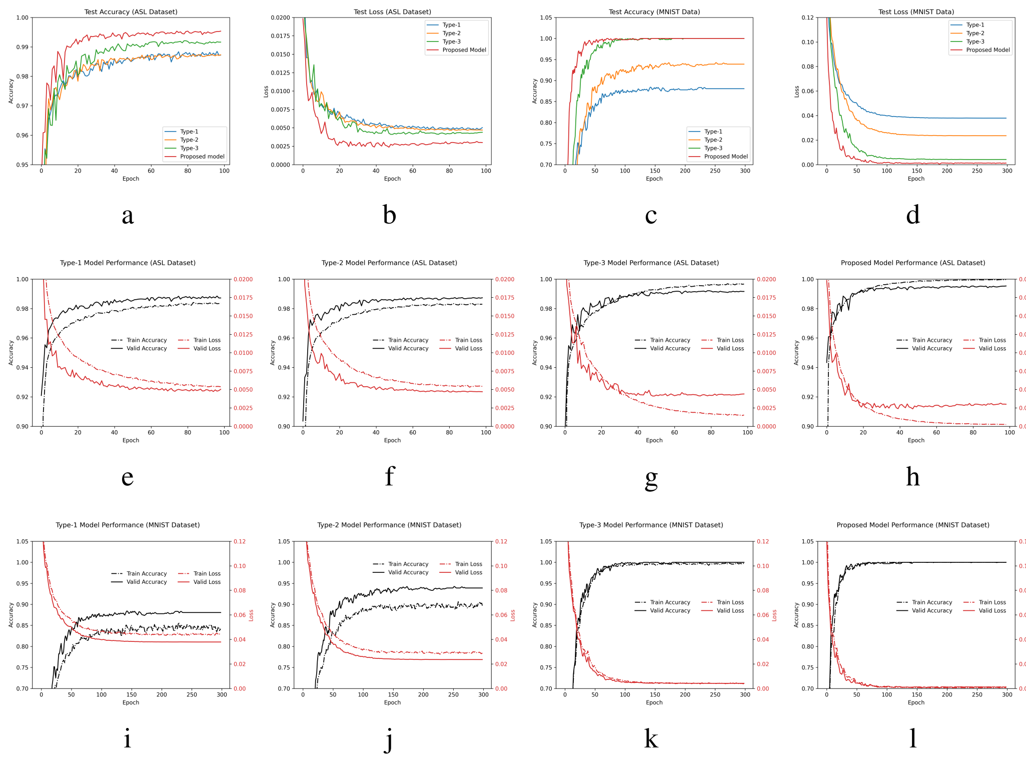


Figure 4.4: Graphs showing Accuracy and Loss metrics for Models 1, 2, 3, and the proposed model on the ASL and MNIST datasets, with (a, b) for ASL dataset accuracy and loss, (c, d) for MNIST dataset accuracy and loss, and (e–h), (i–l) for training and testing performance metrics of each model on ASL and MNIST datasets, respectively.

Table 4.13: Performance comparison of different models

Model	Train accuracy	Test accuracy	Recall	F1-score	Parameters
Type-1	98.32	99.06	99.07	99.06	8601
Type-2	97.99	99.02	99.03	99.02	8893
Type-3	99.79	99.71	99.71	99.71	41833
Proposed	99.95	99.80	99.80	99.80	26793

results (Table 4.13) demonstrate that the models we investigated, particularly the proposed model, achieved outstanding performance in classifying hand gestures.

There were no significant differences in evaluation metrics when conducting experiments on the Vietnamese dataset compared to publicly available datasets. Both models continued to exhibit excellent results on the Vietnamese dataset, with the proposed model maintaining its status as the best-performing model across all evaluation metrics. The accuracy of the proposed model on both the training and test sets was 99.95% and 99.80%, respectively, surpassing the type-3 model with only 99.79% and

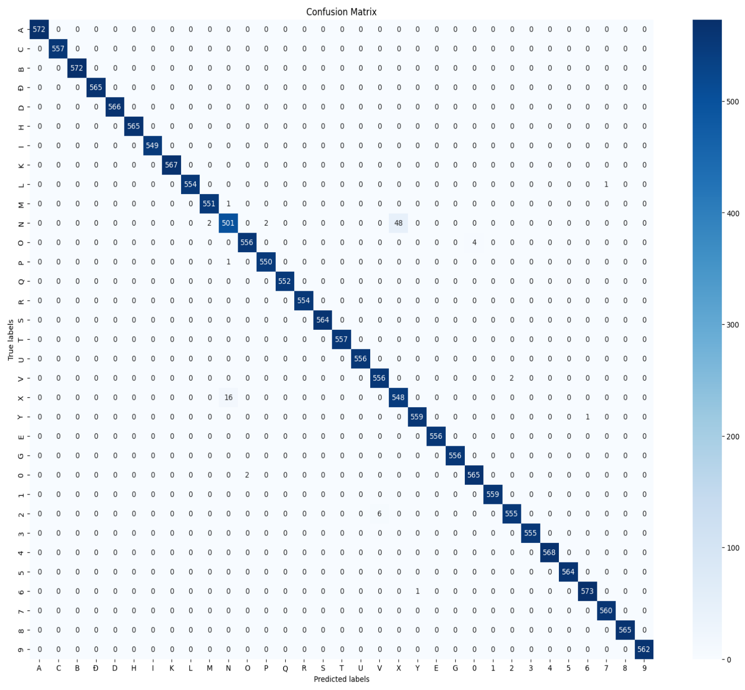


Figure 4.5: Confusion matrix of the proposed model on the test set of the Vietnamese Sign Language (ViSL) dataset.

99.71%. Through the confusion matrix (Figure 4.5), only a minimal amount of data is mislabeled.

From visual observations, we noticed that the Vietnamese sign language gesture dataset and the English sign language gesture dataset share many similarities in the representation of alphabets. Our dataset was meticulously constructed, and its size does not differ significantly from publicly available datasets. Experimental results show that the accuracy of the experimented models is comparable, reinforcing the effectiveness of our research methodology and the quality of the dataset we have constructed.

4.2.4 Conclusion for A New Efficient Optimized Graph Convolutional Neural Network based Multi-Head Attentative for Sign Language Recognition

In this study, we approached this problem using image processing techniques from common devices such as webcams and mobile cameras to recognize signs from various sign languages. However, instead of directly processing raw image data, we employed techniques to estimate the coordinates of hand joints, combined with the connectivity structure of these hand bones to construct data as a graph structure. To enrich graph features with the aim of enhancing model performance, processes such as decision altering of coordinate origins and feature augmentation within each graph node were applied. With this classification model, we expect to apply graph convolution-based models to address the problem of recognizing complex hand shapes. As anticipated, all four models studied, including: (1) the "Basic 2-Layer Graph Convolutional Model consisting of 2 hidden blocks of a graph convolutional network", (2) the "improvement based on model (1) by adding 1 linear layer before the message passing process in the basic GraphConv convolution implementation", (3) the "enhancement of model (2) by combining skip-connection + multi-attention before entering the graph convolution to optimize information processing on the graph", and the proposed model, "a variant of the non-local network with replacing 1x1 convolutional layers with graph convolutional layers to allow local information sharing between components before interacting with other components", all achieved outstanding results for the classification task. Particularly, models type-3 and proposed achieved evaluation metrics above 99.9% on the publicly available ASL datasets and consistently across all metrics. Experimental results also demonstrate that our models yield comparable or superior effectiveness compared to many studies in the same field. A notable point of all four models we studied is their extremely lightweight size, especially the proposed model with the highest accuracy but using only 26793 parameters.

Furthermore, considering the current reality where there are not many carefully constructed and comprehensive datasets available for Vietnamese sign language, we introduced a new dataset called VSL (Vietnamese Sign Language). The dataset comprises over 50,000 images and 33 classes built with diversity and compliance with the guidelines outlined in the official documentation for sign language in Vietnam. The proposed models also exhibited excellent performance on this new dataset, akin to the results obtained when experimenting with publicly available ASL datasets.

We envision that this research will have a significant impact and contribution to the field, serving as a foundation for building more accurate, convenient, and accessible techniques and devices to address challenges for sign language users and communities. In the future, we hope not only to conduct research as experiments in recognizing standard characters but also to expand it to include special characters combined with gestures. At the same time, we aim to develop complete device models that can be easily deployed and accessed by individuals in the community.

4.3 Summary

Summary and Comprehensive Evaluation

In this study, we have demonstrated the effectiveness of Graph-Level Representation Learning in real-world problems through two approaches: applying Graph Convolutional Networks (GCN) to SQL Injection Detection and improving graph-based models to enhance performance in hand recognition tasks. The experiments have shown that deep learning on graphs not only exploits complex relationships among data components but also provides a powerful approach to solving problems with intricate spatial structures.

Effectiveness of Graph Convolutional Networks in SQL Injection Detection

In the first part of our study, we utilized GCN to detect SQL Injection attacks by modeling SQL queries as graphs. Experimental results indicate that GCN outperforms traditional methods such as Random Forest and SVM. Specifically, the model achieved an accuracy of 99%, surpassing baseline models while significantly reducing the false positive rate. This confirms that leveraging relationships among query components enhances classification capability and improves the reliability of attack detection.

One of the key advantages of graph-based models is their superior generalization compared to sequence-based models. Traditional methods often rely on recognizing known attack patterns and struggle with detecting new SQL Injection variants. In contrast, a graph-based approach enables the model to learn the fundamental structure of queries, allowing it to identify both known and previously unseen attack patterns.

Enhancing Graph-Based Models for Hand Recognition

In the second part of our study, we improved the GCN architecture to better suit the hand recognition task. Representing hand skeletons as graphs yielded significant improvements. By integrating spatial and temporal features into a deep learning framework, the model achieved a considerable accuracy boost compared to traditional methods such as CNN and LSTM. Experimental results show that the improved model reached an accuracy of 99.53%.

A crucial enhancement in our model was the incorporation of an attention mechanism to assign importance weights to individual nodes within the graph. This helped the model focus on critical finger joints during gesture recognition, reducing noise and improving generalization. Additionally, the optimized model was designed to lower computational complexity, accelerating inference speed without sacrificing accuracy. This is particularly beneficial when deploying the model on resource-constrained devices.

Potential Applications and Expansion

The findings of this research open up numerous practical applications. The use of deep learning on graphs is not limited to the two problems discussed but can be extended to various fields such as social network analysis, fraud detection in financial transactions, supply chain analysis, and even healthcare, where data can be represented as relational networks.

Furthermore, the graph model improvements proposed in this study can be adapted to suit different types of data. For example, in cybersecurity, rather than focusing solely on SQL Injection, the model could be extended to detect other types of attacks such as XSS or RCE by constructing appropriate graph representations. Similarly, in computer vision, graph-based models could be applied to human gesture recognition or motion analysis in video sequences.

This study has demonstrated the effectiveness of using GCN in real-world applications while proposing significant model enhancements to improve performance. Experimental results indicate that deep learning on graphs not only provides a better representation of complex data relationships but also serves as a powerful and flexible approach for addressing various challenges. Future research directions may focus on further optimizing computational efficiency and exploring unsupervised learning techniques on graphs to reduce dependence on labeled data.

Chapter 5

Conclusion

5.1 Recapitulation

In this study, we proposed and evaluated a deep learning approach based on Graph-Level Representation Learning to enhance Graph-Aware Applications. The proposed model leverages structural information from graphs rather than relying solely on individual node features, thereby improving generalization ability and model performance on real-world tasks.

The core objective of this study is to explore global-level graph representations to optimize the processing capabilities of Deep Neural Networks (DNNs) when applied to complex structured datasets. To demonstrate the effectiveness of this approach, we applied the proposed model to two critical tasks:

- SQL Injection detection by representing SQL queries as graphs.
- Hand gesture recognition using skeletal data.

Experimental results indicate that the proposed method not only improves accuracy but also allows the system to better capture relationships between data components, particularly in non-Euclidean structured data such as graphs.

Specifically, in SQL Injection detection, transforming SQL queries into graph representations enabled Graph Convolutional Networks (GCNs) to extract deeper features, leading to a 99% accuracy rate, significantly outperforming traditional methods. In hand gesture recognition, we enhanced GCNs by integrating Attention Mechanisms and spatiotemporal features, achieving an accuracy of 99.53%.

These findings demonstrate that deep learning on graph-level representations is a promising direction for improving AI system performance in various fields, including network security, behavior recognition, graph data analysis, and computer vision.

5.2 Concluding remarks

Based on the obtained results, we can conclude that integrating Deep Learning with Graph-Level Representation Learning is an effective solution for tasks that require leveraging relational information from graph data.

The proposed model offers three key advantages:

1. Capturing relationships between entities rather than relying solely on individual features. This enables the model to gain a deeper understanding of graph structures, enhancing generalization ability and reducing overfitting.

2. Scalability across multiple applications, including cybersecurity, recommendation systems, social network analysis, and behavior recognition.

3. Performance optimization, achieving higher accuracy compared to traditional methods, especially for tasks involving unstructured or highly interconnected data.

However, despite these advantages, the model still has certain limitations that require further research to enhance its practical applicability.

5.3 Limitations of the research

Although this study has achieved promising results, several important limitations need to be considered in future research:

Preprocessing Requirements for Graph Conversion

One of the main limitations of using deep learning on graphs is the need for a preprocessing step to convert non-graph data into graph form. This conversion can increase computational complexity and require higher processing resources.

For many real-world problems, initial data is often represented in tabular form, sequences, or images without a clear graph structure. Therefore, transformation methods such as constructing relationship networks between entities or extracting features for graph representation must be applied.

This process can involve multiple complex steps, such as defining appropriate nodes and edges, selecting the types of relationships between data components, and determining edge weights if necessary. Furthermore, the construction of a reasonable graph model significantly affects the performance of deep learning models. If the generated graph does not accurately reflect relationships within the data, the model may not achieve optimal performance.

Additionally, converting non-graph data into graph form can substantially increase computational resource requirements, especially for large-scale input data. In systems with limited memory and processing power, this preprocessing step can become a significant challenge. In some cases, creating and storing graphs may result in high memory costs, slowing down overall system performance.

Although leveraging graph models offers clear advantages in capturing relationships within data, the preprocessing requirement can act as a barrier to real-world deployment. Therefore, optimizing the transformation of non-graph data into graphs is crucial, such as employing automatic graph construction algorithms or reducing unnecessary nodes and edges to enhance computational efficiency.

Dependence on Labeled Data

The current model primarily relies on supervised learning, which requires a large amount of labeled data for training. However, in practice, collecting and labeling graph data is often expensive and time-consuming.

As a result, this approach is not yet effectively applicable to large datasets with insufficient labels, such as social networks or security monitoring systems.

Generalization Ability Across Various Graph Data Types

Although the model has demonstrated effectiveness in two specific tasks (SQL Injection detection and hand recognition), further testing on diverse datasets is needed to assess its generalization capability.

Some applications require processing dynamic graphs, but this study has focused only on static graphs.

5.4 Suggestions for further research

To overcome the above limitations and expand the applications of deep learning in graph representations, we propose several future research directions:

Developing Unsupervised Learning Methods for Graphs

- Utilize Self-Supervised Learning to learn graph representations without requiring extensive labeled data.
- Incorporate Contrastive Learning to enhance feature extraction from unlabeled graphs.

Extending Models to Dynamic Graphs and Temporal Data

- Investigate Graph Neural Networks for dynamic graphs (Dynamic GNNs) to handle evolving systems such as social networks, traffic systems, and financial data.
- Combine graph learning with temporal data (Graph-Temporal Learning) to address time-dependent applications, such as financial fraud detection and user behavior analysis.

Applying Graph Learning to Other Real-World Problems

- Develop recommendation systems based on global graph representations.
- Apply GNNs to malware detection by representing software programs as graphs and classifying them accordingly.

References

[1] Duc-Chinh Nguyen, Manh-Hung Ha, and Oscal Tzyh-Chiang Chen. “Towards Lightweight Model Using Non-local-based Graph Convolution Neural Network for SQL Injection Detection”. In: *The Egyptian Informatics Journal* (). Revision.

[2] Duc-Chinh Nguyen, Manh-Hung Ha, and Oscal Tzyh-Chiang Chen. “A New Efficient Optimized Graph Convolutional Neural Network based Multi-Head Attentative for Sign Language Recognition”. In: *The Egyptian Informatics Journal* (). Sumitted.

[3] Duc-Chinh Nguyen et al. “RHM: Novel Graph Convolution Based on Non-Local Network for SQL Injection Identification”. In: *2024 IEEE Symposium on Industrial Electronics amp;amp; Applications (ISIEA)*. IEEE, July 2024, pp. 1–5. DOI: 10.1109/isiea61920.2024.10607303. URL: http://dx.doi.org/10.1109/isiea61920.2024.10607303.

[4] Duc-Chinh Nguyen et al. “Lightweight Graph Convolutional Network Based on Multi-Head Residual Attention for Hand Point Classification”. In: *2024 IEEE International Conference on Visual Communications and Image Processing (VCIP)*. IEEE, Dec. 2024, pp. 1–5. DOI: 10.1109/vcip63160.2024.10849902. URL: http://dx.doi.org/10.1109/vcip63160.2024.10849902.

[5] Duc-Chinh Nguyen, Manh-Hung Ha, and Long-Quang-Anh Tran. “Enhancing Physical Rehabilitation Evaluation with Temporal Graph Convolution Networks”. In: *5th International Conference on Intel- ligent Systems Networks* (). Accepted.

[6] Manh-Tuan Do et al. “Toward improving precision and complexity of transformer-based cost-sensitive learning models for plant disease detection”. In: *Frontiers in Computer Science* 6 (Jan. 2025). ISSN: 2624-9898. DOI: 10.3389/fcomp.2024.1480481. URL: http://dx.doi.org/10.3389/fcomp.2024.1480481.

[7] Manh-Hung Ha et al. “Emotional Inference from Speech Signals Informed by Multiple Stream DNNs Based Non-Local Attention Mechanism”. In: *EAI Endorsed Transactions on Industrial Networks and Intelligent Systems* 11.4 (Aug. 2024). ISSN: 2410-0218. DOI: 10.4108/eetinis.v11i4.4734. URL: http://dx.doi.org/10.4108/eetinis.v11i4.4734.

[8] Manh-Hung Ha et al. “Plant pathology identification using local-global feature level based on transformer”. In: *Indonesian Journal of Electrical Engineering and Computer Science* 34.3 (June 2024), p. 1582. ISSN: 2502-4752. DOI: 10.11591/ijeecs.v34.i3.pp1582-1592. URL: http://dx.doi.org/10.11591/ijeecs.v34.i3.pp1582-1592.

[9] Manh-Hung Ha, Duc-Chinh Nguyen, and Minh-Huy Le. “Low-High Feature Based Local-Global Attention for Traffic Police Action Identification”. In: *2023 Asia Meeting on Environment and Electrical Engineering (EEE-AM)*. IEEE, Nov. 2023, pp. 1–4. DOI: 10.1109/eee-am58328.2023.10395832. URL: http://dx.doi.org/10.1109/eee-am58328.2023.10395832.

[10] Manh-Tuan Do et al. “You Only Look Once Based-C2fGhost Using Efficient SIoU Loss Function for Airplane Detection”. In: *2024 9th International Conference on Frontiers of Signal Processing (ICFSP)*. IEEE, Sept. 2024, pp. 1–5. DOI: 10.1109/icfsp62546.2024.10785456. URL: http://dx.doi.org/10.1109/icfsp62546.2024.10785456.

[11] Manh-Tuan Do et al. “Human Detection Based Yolo Backbones-Transformer in UAVs”. In: *2023 International Conference on System Science and Engineering (ICSSE)*. IEEE, July 2023, pp. 576–580. DOI: 10.1109/icsse58758.2023.10227141. URL: http://dx.doi.org/10.1109/icsse58758.2023.10227141.

[12] Manh-Tuan Do et al. “An Effective Method for Detecting Personal Protective Equipment at Real Construction Sites Using the Improved YOLOv5s with SIoU Loss Function”. In: *2023 RIVF International Conference on Computing and Communication Technologies (RIVF)*. IEEE, Dec. 2023, pp. 430–434. DOI: 10.1109/rivf60135.2023.10471799. URL: http://dx.doi.org/10.1109/rivf60135.2023.10471799.

[13] N.D Quang Anh et al. “VNEMOS: Vietnamese Speech Emotion Inference Using Deep Neural Networks”. In: *2024 9th International Conference on Integrated Circuits, Design, and Verification (ICDV)*. IEEE, June 2024, pp. 97–101. DOI: 10.1109/icdv61346.2024.10616411. URL: http://dx.doi.org/10.1109/icdv61346.2024.10616411.

[14] A. Sperduti and A. Starita. “Supervised neural networks for the classification of structures”. In: *IEEE Transactions on Neural Networks* 8.3 (May 1997), pp. 714–735. ISSN: 1941-0093. DOI: 10.1109/72.572108. URL: http://dx.doi.org/10.1109/72.572108.

[15] M. Gori, G. Monfardini, and F. Scarselli. "A new model for learning in graph domains". In: *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.* Vol. 2. IJCNN-05. IEEE, pp. 729–734. DOI: 10.1109/ijcnn.2005.1555942. URL: http://dx.doi.org/10.1109/ijcnn.2005.1555942.

[16] F. Scarselli et al. "The Graph Neural Network Model". In: *IEEE Transactions on Neural Networks* 20.1 (Jan. 2009), pp. 61–80. ISSN: 1941-0093. DOI: 10.1109/tnn.2008.2005605. URL: http://dx.doi.org/10.1109/tnn.2008.2005605.

[17] Claudio Gallicchio and Alessio Micheli. "Graph Echo State Networks". In: *The 2010 International Joint Conference on Neural Networks (IJCNN)*. IEEE, July 2010, pp. 1–8. DOI: 10.1109/ijcnn.2010.5596796. URL: http://dx.doi.org/10.1109/ijcnn.2010.5596796.

[18] Yujia Li et al. "Gated graph sequence neural networks". In: *arXiv preprint arXiv:1511.05493* (2015).

[19] Hanjun Dai et al. "Learning steady-states of iterative algorithms over graphs". In: *International conference on machine learning*. PMLR. 2018, pp. 1106–1114.

[20] Joan Bruna et al. "Spectral networks and locally connected networks on graphs". In: *arXiv preprint arXiv:1312.6203* (2013).

[21] Mikael Henaff, Joan Bruna, and Yann LeCun. "Deep convolutional networks on graph-structured data". In: *arXiv preprint arXiv:1506.05163* (2015).

[22] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. "Convolutional neural networks on graphs with fast localized spectral filtering". In: *Advances in neural information processing systems* 29 (2016).

[23] Thomas N Kipf and Max Welling. "Semi-supervised classification with graph convolutional networks". In: *arXiv preprint arXiv:1609.02907* (2016).

[24] Ron Levie et al. "CayleyNets: Graph Convolutional Neural Networks With Complex Rational Spectral Filters". In: *IEEE Transactions on Signal Processing* 67.1 (Jan. 2019), pp. 97–109. ISSN: 1941-0476. DOI: 10.1109/tsp.2018.2879624. URL: http://dx.doi.org/10.1109/tsp.2018.2879624.

[25] A. Micheli. "Neural Network for Graphs: A Contextual Constructive Approach". In: *IEEE Transactions on Neural Networks* 20.3 (Mar. 2009), pp. 498–511. ISSN: 1941-0093. DOI: 10.1109/tnn.2008.2010350. URL: http://dx.doi.org/10.1109/tnn.2008.2010350.

[26] James Atwood and Don Towsley. “Diffusion-convolutional neural networks”. In: *Advances in neural information processing systems* 29 (2016).

[27] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. “Learning convolutional neural networks for graphs”. In: *International conference on machine learning*. PMLR. 2016, pp. 2014–2023.

[28] Justin Gilmer et al. “Neural message passing for quantum chemistry”. In: *International conference on machine learning*. PMLR. 2017, pp. 1263–1272.

[29] William L Hamilton, Rex Ying, and Jure Leskovec. “Representation learning on graphs: Methods and applications”. In: *arXiv preprint arXiv:1709.05584* (2017).

[30] Peng Cui et al. “A Survey on Network Embedding”. In: *IEEE Transactions on Knowledge and Data Engineering* 31.5 (May 2019), pp. 833–852. ISSN: 2326-3865. DOI: 10.1109/tkde.2018.2849727. URL: http://dx.doi.org/10.1109/tkde.2018.2849727.

[31] Daokun Zhang et al. “Network Representation Learning: A Survey”. In: *IEEE Transactions on Big Data* 6.1 (Mar. 2020), pp. 3–28. ISSN: 2372-2096. DOI: 10.1109/tbdata.2018.2850013. URL: http://dx.doi.org/10.1109/tbdata.2018.2850013.

[32] Hongyun Cai, Vincent W. Zheng, and Kevin Chen-Chuan Chang. “A Comprehensive Survey of Graph Embedding: Problems, Techniques, and Applications”. In: *IEEE Transactions on Knowledge and Data Engineering* 30.9 (Sept. 2018), pp. 1616–1637. ISSN: 2326-3865. DOI: 10.1109/tkde.2018.2807452. URL: http://dx.doi.org/10.1109/tkde.2018.2807452.

[33] Palash Goyal and Emilio Ferrara. “Graph embedding techniques, applications, and performance: A survey”. In: *Knowledge-Based Systems* 151 (July 2018), pp. 78–94. ISSN: 0950-7051. DOI: 10.1016/j.knosys.2018.03.022. URL: http://dx.doi.org/10.1016/j.knosys.2018.03.022.

[34] Shirui Pan et al. “Tri-party deep network representation”. In: *25th International Joint Conference on Artificial Intelligence (IJCAI-16)*. AAAI Press/International Joint Conferences on Artificial Intelligence. 2016.

[35] Xiaobo Shen et al. “Discrete network embedding”. In: *27th International Joint Conference on Artificial Intelligence (IJCAI)*. International Joint Conference on Artificial Intelligence (IJCAI). 2018.

[36] Hong Yang et al. “Binarized attributed network embedding”. In: *2018 IEEE International Conference on Data Mining (ICDM)*. IEEE, Nov. 2018, pp. 1476–1481. DOI: 10.1109/icdm.2018.8626170. URL: http://dx.doi.org/10.1109/icdm.2018.8626170.

[37] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. “Deepwalk: Online learning of social representations”. In: *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2014, pp. 701–710.

[38] S Vichy N Vishwanathan et al. “Graph kernels”. In: *The Journal of Machine Learning Research* 11 (2010), pp. 1201–1242.

[39] Nino Shervashidze et al. “Weisfeiler-lehman graph kernels.” In: *Journal of Machine Learning Research* 12.9 (2011).

[40] Nicolò Navarin, Alessandro Sperduti, et al. “Approximated Neighbours MinHash Graph Node Kernel.” In: *ESANN*. 2017, pp. 281–286.

[41] Nils M Kriege, Fredrik D Johansson, and Christopher Morris. “A survey on graph kernels”. In: *Applied Network Science* 5 (2020), pp. 1–42.

[42] D. I. Shuman et al. “The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains”. In: *IEEE Signal Processing Magazine* 30.3 (May 2013), pp. 83–98. ISSN: 1053-5888. DOI: 10.1109/msp.2012.2235192. URL: http://dx.doi.org/10.1109/msp.2012.2235192.

[43] Aliaksei Sandryhaila and Jose M. F. Moura. “Discrete signal processing on graphs: Graph filters”. In: *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, May 2013. DOI: 10.1109/icassp.2013.6638849. URL: http://dx.doi.org/10.1109/icassp.2013.6638849.

[44] Siheng Chen et al. “Discrete Signal Processing on Graphs: Sampling Theory”. In: *IEEE Transactions on Signal Processing* 63.24 (Dec. 2015), pp. 6510–6523. ISSN: 1941-0476. DOI: 10.1109/tsp.2015.2469645. URL: http://dx.doi.org/10.1109/tsp.2015.2469645.

[45] *OWASP Top 10 - 2021 The Ten Most Critical Web Application Security Risks*. Accessed. URL: https://owasp.org/Top10/.

[46] Ines Jemal et al. “Sql injection attack detection and prevention techniques using machine learning”. In: *International Journal of Applied Engineering Research* 15.6 (2020), pp. 569–580.

[47] *Sony Says PlayStation Hacker Got Personal Data*. Accessed. URL: https://www.nytimes.com/2011/04/27/technology/27playstation.html.

[48] Swapnil Kharche, Kanchan Gohad, Bharti Ambetkar, et al. "Preventing SQL injection attack using pattern matching algorithm". In: *arXiv preprint arXiv:1504.06920* (2015).

[49] Fahim K. Sufi, Musleh Alsulami, and Adnan Gutub. "Automating Global Threat-Maps Generation via Advancements of News Sensors and AI". In: *Arabian Journal for Science and Engineering* 48.2 (Oct. 2022), pp. 2455–2472. ISSN: 2191-4281. DOI: 10.1007/s13369-022-07250-1. URL: http://dx.doi.org/10.1007/s13369-022-07250-1.

[50] William G. J. Halfond and Alessandro Orso. "AMNESIA: analysis and monitoring for NEutralizing SQL-injection attacks". In: *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering*. ASE05. ACM, Nov. 2005, pp. 174–183. DOI: 10.1145/1101908.1101935. URL: http://dx.doi.org/10.1145/1101908.1101935.

[51] Nuno Antunes and Marco Vieira. "Detecting SQL Injection Vulnerabilities in Web Services". In: *2009 Fourth Latin-American Symposium on Dependable Computing*. IEEE, Sept. 2009, pp. 17–24. DOI: 10.1109/ladc.2009.21. URL: http://dx.doi.org/10.1109/ladc.2009.21.

[52] Sahar Altalhi and Adnan Gutub. "A survey on predictions of cyber-attacks utilizing real-time twitter tracing recognition". In: *Journal of Ambient Intelligence and Humanized Computing* 12.11 (Jan. 2021), pp. 10209–10221. ISSN: 1868-5145. DOI: 10.1007/s12652-020-02789-z. URL: http://dx.doi.org/10.1007/s12652-020-02789-z.

[53] Afnan Mahmud Al Badri and Sahel Alouneh. "Detection of Malicious Requests to Protect Web Applications and DNS Servers Against SQL Injection Using Machine Learning". In: *2023 International Conference on Intelligent Computing, Communication, Networking and Services (ICCNS)*. IEEE, June 2023, pp. 5–11. DOI: 10.1109/iccns58795.2023.10193085. URL: http://dx.doi.org/10.1109/iccns58795.2023.10193085.

[54] Eman Hosam et al. "SQL Injection Detection Using Machine Learning Techniques". In: *2021 8th International Conference on Soft Computing amp; Machine Intelligence (ISCMI)*. IEEE, Nov. 2021. DOI: 10.1109/iscmi53840.2021.9654820. URL: http://dx.doi.org/10.1109/iscmi53840.2021.9654820.

[55] Aidana Zhumabekova et al. “Determining Web Application Vulnerabilities Using Machine Learning Methods”. In: *2023 19th International Asian School-Seminar on Optimization Problems of Complex Systems (OPCS)*. IEEE, Aug. 2023, pp. 136–139. DOI: 10.1109/opcs59592.2023.10275756. URL: http://dx.doi.org/10.1109/opcs59592.2023.10275756.

[56] Kasim Tasdemir et al. “An Investigation of Machine Learning Algorithms for High-bandwidth SQL Injection Detection Utilising BlueField-3 DPU Technology”. In: *2023 IEEE 36th International System-on-Chip Conference (SOCC)*. IEEE, Sept. 2023, pp. 1–6. DOI: 10.1109/socc58585.2023.10256777. URL: http://dx.doi.org/10.1109/socc58585.2023.10256777.

[57] Stanislav Abaimov and Giuseppe Bianchi. “CODDLE: Code-Injection Detection With Deep Learning”. In: *IEEE Access* 7 (2019), pp. 128617–128627. ISSN: 2169-3536. DOI: 10.1109/access.2019.2939870. URL: http://dx.doi.org/10.1109/access.2019.2939870.

[58] Nisrean Thalji et al. “AE-Net: Novel Autoencoder-Based Deep Features for SQL Injection Attack Detection”. In: *IEEE Access* 11 (2023), pp. 135507–135516. ISSN: 2169-3536. DOI: 10.1109/access.2023.3337645. URL: http://dx.doi.org/10.1109/access.2023.3337645.

[59] Kevin Zhang. “A Machine Learning Based Approach to Identify SQL Injection Vulnerabilities”. In: *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, Nov. 2019, pp. 1286–1288. DOI: 10.1109/ase.2019.00164. URL: http://dx.doi.org/10.1109/ase.2019.00164.

[60] Rémi Lebret and Ronan Collobert. “Word Embeddings through Hellinger PCA”. In: *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics, 2014. DOI: 10.3115/v1/e14-1051. URL: http://dx.doi.org/10.3115/v1/e14-1051.

[61] Tomas Mikolov et al. “Efficient estimation of word representations in vector space”. In: *arXiv preprint arXiv:1301.3781* (2013).

[62] Xin Xie et al. “SQL Injection Detection for Web Applications Based on Elastic-Pooling CNN”. In: *IEEE Access* 7 (2019), pp. 151475–151481. ISSN: 2169-3536. DOI: 10.1109/access.2019.2947527. URL: http://dx.doi.org/10.1109/access.2019.2947527.

[63] Gowtham M and Pramod H B. “Semantic Query-Featured Ensemble Learning Model for SQL-Injection Attack Detection in IoT-Ecosystems”. In: *IEEE Transactions on Reliability* 71.2 (June 2022), pp. 1057–1074. ISSN: 1558-1721. DOI: 10.1109/tr.2021.3124331. URL: http://dx.doi.org/10.1109/tr.2021.3124331.

[64] Ao Luo, Wei Huang, and Wenqing Fan. “A CNN-based Approach to the Detection of SQL Injection Attacks”. In: *2019 IEEE/ACIS 18th International Conference on Computer and Information Science (ICIS)*. IEEE, June 2019. DOI: 10.1109/icis46139.2019.8940196. URL: http://dx.doi.org/10.1109/icis46139.2019.8940196.

[65] Peng Tang et al. “SQL Injection Behavior Mining Based Deep Learning”. In: *Advanced Data Mining and Applications*. Springer International Publishing, 2018, pp. 445–454. ISBN: 9783030050900. DOI: 10.1007/978-3-030-05090-0_38. URL: http://dx.doi.org/10.1007/978-3-030-05090-0_38.

[66] Qi Li et al. “LSTM-based SQL Injection Detection Method for Intelligent Transportation System”. In: *IEEE Transactions on Vehicular Technology* (2019), pp. 1–1. ISSN: 1939-9359. DOI: 10.1109/tvt.2019.2893675. URL: http://dx.doi.org/10.1109/tvt.2019.2893675.

[67] Pengcheng Wen et al. “SQL Injection Detection Technology Based on BiLSTM-Attention”. In: *2021 4th International Conference on Robotics, Control and Automation Engineering (RCAE)*. IEEE, Nov. 2021, pp. 165–170. DOI: 10.1109/rcae53607.2021.9638837. URL: http://dx.doi.org/10.1109/rcae53607.2021.9638837.

[68] Ashish Vaswani et al. “Attention is all you need”. In: *Advances in neural information processing systems* 30 (2017).

[69] Jacob Devlin et al. “Bert: Pre-training of deep bidirectional transformers for language understanding”. In: *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*. 2019, pp. 4171–4186.

[70] Ashish Singh et al. “AI-Based Mobile Edge Computing for IoT: Applications, Challenges, and Future Scope”. In: *Arabian Journal for Science and Engineering* 47.8 (Jan. 2022), pp. 9801–9831. ISSN: 2191-4281. DOI: 10.1007/s13369-021-06348-2. URL: http://dx.doi.org/10.1007/s13369-021-06348-2.

[71] Piotr Bojanowski et al. “Enriching Word Vectors with Subword Information”. In: *Transactions of the Association for Computational Linguistics* 5 (Dec. 2017), pp. 135–146. ISSN: 2307-387X. DOI: 10.1162/tacl_a_00051. URL: http://dx.doi.org/10.1162/tacl_a_00051.

[72] Yiming Cui et al. “Attention-over-Attention Neural Networks for Reading Comprehension”. In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, 2017. DOI: 10.18653/v1/p17-1055. URL: http://dx.doi.org/10.18653/v1/p17-1055.

[73] Jan Chorowski et al. “End-to-end continuous speech recognition using attention-based recurrent NN: First results”. In: *arXiv preprint arXiv:1412.1602* (2014).

[74] Manh-Hung Ha and Oscal Tzyh-Chiang Chen. “Non-Local Spatiotemporal Correlation Attention for Action Recognition”. In: *2022 IEEE International Conference on Multimedia and Expo Workshops (ICMEW)*. IEEE, July 2022, pp. 1–4. DOI: 10.1109/icmew56448.2022.9859314. URL: http://dx.doi.org/10.1109/icmew56448.2022.9859314.

[75] *SQL Style Guide*. Accessed. URL: https://www.sqlstyle.guide/.

[76] Keyulu Xu et al. “Representation learning on graphs with jumping knowledge networks”. In: *International conference on machine learning*. PMLR. 2018, pp. 5453–5462.

[77] Uri Alon and Eran Yahav. “On the bottleneck of graph neural networks and its practical implications”. In: *arXiv preprint arXiv:2006.05205* (2020).

[78] World Health Organization. *Deafness and hearing loss*. Accessed. URL: https://www.who.int/news-room/fact-sheets/detail/deafness-and-hearing-loss.

[79] United Nations. *Sign languages unite us!* Accessed. URL: https://www.un.org/en/observances/sign-languages-day.

[80] Simin Yuan et al. “Chinese Sign Language Alphabet Recognition Based on Random Forest Algorithm”. In: *2020 IEEE International Workshop on Metrology for Industry 4.0 amp; IoT*. IEEE, June 2020, pp. 340–344. DOI: 10.1109/metroind4.0iot48571.2020.9138285. URL: http://dx.doi.org/10.1109/metroind4.0iot48571.2020.9138285.

[81] Cao Dong, Ming C Leu, and Zhaozheng Yin. “American sign language alphabet recognition using microsoft kinect”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*. 2015, pp. 44–52.

[82] Watcharin Tangsuksant, Suchin Adhan, and Chuchart Pintavirooj. “American Sign Language recognition by using 3D geometric invariant feature and ANN classification”. In: *The 7th 2014 Biomedical Engineering International Conference*. IEEE, Nov. 2014. DOI: 10.1109/bmeicon.2014.7017372. URL: http://dx.doi.org/10.1109/bmeicon.2014.7017372.

[83] Salem Ameen and Sunil Vadera. “A convolutional neural network to classify American Sign Language fingerspelling from depth and colour images”. In: *Expert Systems* 34.3 (Feb. 2017). ISSN: 1468-0394. DOI: 10.1111/exsy.12197. URL: http://dx.doi.org/10.1111/exsy.12197.

[84] Jungpil Shin et al. “American Sign Language Alphabet Recognition by Extracting Feature from Hand Pose Estimation”. In: *Sensors* 21.17 (Aug. 2021), p. 5856. ISSN: 1424-8220. DOI: 10.3390/s21175856. URL: http://dx.doi.org/10.3390/s21175856.

[85] Dewinta Aryanie and Yaya Heryadi. “American sign language-based fingerspelling recognition using k-Nearest Neighbors classifier”. In: *2015 3rd International Conference on Information and Communication Technology (ICoICT)*. IEEE, May 2015, pp. 533–536. DOI: 10.1109/icoict.2015.7231481. URL: http://dx.doi.org/10.1109/icoict.2015.7231481.

[86] Cheok Ming Jin, Zaid Omar, and Mohamed Hisham Jaward. “A mobile application of American sign language translation via image processing algorithms”. In: *2016 IEEE Region 10 Symposium (TENSYMP)*. IEEE, May 2016, pp. 104–109. DOI: 10.1109/tenconspring.2016.7519386. URL: http://dx.doi.org/10.1109/tenconspring.2016.7519386.

[87] Tse-Yu Pan et al. “Real-Time Sign Language Recognition in Complex Background Scene Based on a Hierarchical Clustering Classification Method”. In: *2016 IEEE Second International Conference on Multimedia Big Data (BigMM)*. IEEE, Apr. 2016, pp. 64–67. DOI: 10.1109/bigmm.2016.44. URL: http://dx.doi.org/10.1109/bigmm.2016.44.

[88] Md. Mehedi Hasan et al. “Classification of Sign Language Characters by Applying a Deep Convolutional Neural Network”. In: *2020 2nd International Conference on Advanced Information and Communication Technology (ICAICT)*.

IEEE, Nov. 2020, pp. 434–438. DOI: 10.1109/icaict51780.2020.9333456. URL: http://dx.doi.org/10.1109/icaict51780.2020.9333456.

[89] Lean Karlo S Tolentino et al. "Static sign language recognition using deep learning". In: *International Journal of Machine Learning and Computing* 9.6 (2019), pp. 821–827.

[90] Lamptey K. Odartey et al. "Ghanaian Sign Language Recognition Using Deep Learning". In: *Proceedings of the 2019 the International Conference on Pattern Recognition and Artificial Intelligence*. PRAI '19. ACM, Aug. 2019, pp. 81–86. DOI: 10.1145/3357777.3357784. URL: http://dx.doi.org/10.1145/3357777.3357784.

[91] Mayand Kumar et al. "Sign Language Alphabet Recognition Using Convolution Neural Network". In: *2021 5th International Conference on Intelligent Computing and Control Systems (ICICCS)*. IEEE, May 2021, pp. 1859–1865. DOI: 10.1109/iciccs51141.2021.9432296. URL: http://dx.doi.org/10.1109/iciccs51141.2021.9432296.

[92] Vanita Jain et al. "American Sign Language recognition using Support Vector Machine and Convolutional Neural Network". In: *International Journal of Information Technology* 13.3 (Feb. 2021), pp. 1193–1200. ISSN: 2511-2112. DOI: 10.1007/s41870-021-00617-x. URL: http://dx.doi.org/10.1007/s41870-021-00617-x.

[93] Rajesh George Rajan and M. Judith Leo. "American Sign Language Alphabets Recognition using Hand Crafted and Deep Learning Features". In: *2020 International Conference on Inventive Computation Technologies (ICICT)*. IEEE, Feb. 2020, pp. 430–434. DOI: 10.1109/icict48043.2020.9112481. URL: http://dx.doi.org/10.1109/icict48043.2020.9112481.

[94] Huy B.D Nguyen and Hung Ngoc Do. "Deep Learning for American Sign Language Fingerspelling Recognition System". In: *2019 26th International Conference on Telecommunications (ICT)*. IEEE, Apr. 2019, pp. 314–318. DOI: 10.1109/ict.2019.8798856. URL: http://dx.doi.org/10.1109/ict.2019.8798856.

[95] Md Asif Jalal et al. "American Sign Language Posture Understanding with Deep Neural Networks". In: *2018 21st International Conference on Information Fusion (FUSION)*. IEEE, July 2018, pp. 573–579. DOI: 10.23919/icif.2018.8455725. URL: http://dx.doi.org/10.23919/icif.2018.8455725.

[96] Neel Kamal Bhagat, Y. Vishnusai, and G. N. Rathna. “Indian Sign Language Gesture Recognition using Image Processing and Deep Learning”. In: *2019 Digital Image Computing: Techniques and Applications (DICTA)*. IEEE, Dec. 2019, pp. 1–8. DOI: 10.1109/dicta47822.2019.8945850. URL: http://dx.doi.org/10.1109/dicta47822.2019.8945850.

[97] Saleh Aly et al. “Arabic sign language fingerspelling recognition from depth and intensity images”. In: *2016 12th International Computer Engineering Conference (ICENCO)*. IEEE, Dec. 2016, pp. 99–104. DOI: 10.1109/icenco.2016.7856452. URL: http://dx.doi.org/10.1109/icenco.2016.7856452.

[98] Duc-Chinh Nguyen et al. “Lightweight Graph Convolutional Network Based on Multi-Head Residual Attention for Hand Point Classification”. In: *2024 IEEE International Conference on Visual Communications and Image Processing (VCIP)*. IEEE, Dec. 2024, pp. 1–5. DOI: 10.1109/vcip63160.2024.10849902. URL: http://dx.doi.org/10.1109/vcip63160.2024.10849902.

[99] Manuel Eugenio Morocho Cayamcela and Wansu Lim. “Fine-tuning a pre-trained Convolutional Neural Network Model to translate American Sign Language in Real-time”. In: *2019 International Conference on Computing, Networking and Communications (ICNC)*. IEEE, Feb. 2019, pp. 100–104. DOI: 10.1109/iccnc.2019.8685536. URL: http://dx.doi.org/10.1109/iccnc.2019.8685536.

[100] Bader Alsharif et al. “Deep Learning Technology to Recognize American Sign Language Alphabet”. In: *Sensors* 23.18 (Sept. 2023), p. 7970. ISSN: 1424-8220. DOI: 10.3390/s23187970. URL: http://dx.doi.org/10.3390/s23187970.

[101] Helcy D. Alon et al. “Deep-Hand: A Deep Inference Vision Approach of Recognizing a Hand Sign Language using American Alphabet”. In: *2021 International Conference on Computational Intelligence and Knowledge Economy (ICCIKE)*. IEEE, Mar. 2021, pp. 373–377. DOI: 10.1109/iccike51210.2021.9410803. URL: http://dx.doi.org/10.1109/iccike51210.2021.9410803.

[102] Xiaolong Wang et al. “Non-local Neural Networks”. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. IEEE, June 2018. DOI:

10.1109/cvpr.2018.00813. URL: http://dx.doi.org/10.1109/cvpr.2018.00813.

[103] Camillo Lugaresi et al. “Mediapipe: A framework for building perception pipelines”. In: *arXiv preprint arXiv:1906.08172* (2019).

[104] Fan Zhang et al. “Mediapipe hands: On-device real-time hand tracking”. In: *arXiv preprint arXiv:2006.10214* (2020).

[105] Christopher Morris et al. “Weisfeiler and Leman Go Neural: Higher-Order Graph Neural Networks”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 33.01 (July 2019), pp. 4602–4609. ISSN: 2159-5399. DOI: 10.1609/aaai.v33i01.33014602. URL: http://dx.doi.org/10.1609/aaai.v33i01.33014602.

[106] S. S. H. Shah. *SQL Injection Dataset*. [Online]. URL: https://www.kaggle.com/datasets/syedsaqlainhussain/sql-injection-dataset.

[107] M. S. Abu Syeed Sajid Ahmed. *SQL Injection Dataset*. [Online]. URL: https://www.kaggle.com/datasets/sajid576/sql-injection-dataset.

[108] Kasim Tasdemir et al. “Advancing SQL Injection Detection for High-Speed Data Centers: A Novel Approach Using Cascaded NLP”. In: *arXiv preprint arXiv:2312.13041* (2023).

[109] Akash. *ASL Alphabet*. [Online]. URL: https://www.kaggle.com/datasets/grassknoted/asl-alphabet.

[110] *Sign Language MNIST*. [Online]. URL: https://www.kaggle.com/datasets/datamunge/sign-language-mnist.

[111] WenYa Yu. *Biggest SQL Injection Dataset*. [Online]. URL: https://www.kaggle.com/datasets/gambleryu/biggest-sql-injection-dataset.

[112] *SQL-Injection-Extend*. [Online]. URL: https://www.kaggle.com/datasets/alextrinity/sqlinjectionextend.

[113] Ayah Khaldi and Amani Krieshan. *SQL Injection dataset*. [Online]. URL: https://www.kaggle.com/datasets/ayahkhaldi/sql-injection-dataset.

[114] *SQL Injection Dataset*. [Online]. URL: https://www.kaggle.com/datasets/rayten/sql-injection-dataset.

[115] Batool Yahya AlKhuraym, Mohamed Maher Ben Ismail, and Ouiem Bchir. "Arabic Sign Language Recognition using Lightweight CNN-based Architecture". In: *International Journal of Advanced Computer Science and Applications* 13.4 (2022). ISSN: 2158-107X. DOI: 10.14569/ijacsa.2022.0130438. URL: http://dx.doi.org/10.14569/ijacsa.2022.0130438.

[116] Ying Ma, Tianpei Xu, and Kangchul Kim. "Two-Stream Mixed Convolutional Neural Network for American Sign Language Recognition". In: *Sensors* 22.16 (Aug. 2022), p. 5959. ISSN: 1424-8220. DOI: 10.3390/s22165959. URL: http://dx.doi.org/10.3390/s22165959.

[117] Eshed Ohn-Bar and Mohan Manubhai Trivedi. "Hand Gesture Recognition in Real Time for Automotive Interfaces: A Multimodal Vision-Based Approach and Evaluations". In: *IEEE Transactions on Intelligent Transportation Systems* 15.6 (Dec. 2014), pp. 2368–2377. ISSN: 1558-0016. DOI: 10.1109/tits.2014.2337331. URL: http://dx.doi.org/10.1109/tits.2014.2337331.

[118] Michal Lech, Bozena Kostek, and Andrzej Czyżewski. "Examining Classifiers Applied to Static Hand Gesture Recognition in Novel Sound Mixing System". In: *Multimedia and Internet Systems: Theory and Practice*. Springer Berlin Heidelberg, 2013, pp. 77–86. ISBN: 9783642323355. DOI: 10.1007/978-3-642-32335-5_8. URL: http://dx.doi.org/10.1007/978-3-642-32335-5_8.

[119] Debasrita Chakraborty et al. "Trigger Detection System for American Sign Language using Deep Convolutional Neural Networks". In: *Proceedings of the 10th International Conference on Advances in Information Technology*. IAIT 2018. ACM, Dec. 2018, pp. 1–6. DOI: 10.1145/3291280.3291783. URL: http://dx.doi.org/10.1145/3291280.3291783.

[120] Metin Bilgin and Korhan Mutludogan. "American Sign Language Character Recognition with Capsule Networks". In: *2019 3rd International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT)*. IEEE, Oct. 2019, pp. 1–6. DOI: 10.1109/ismsit.2019.8932829. URL: http://dx.doi.org/10.1109/ismsit.2019.8932829.