TOPOLOGY-AWARE CORRECTION OF RETINAL VESSEL GRAPHS USING GRAPH NEURAL NETWORKS

by

FATIMA MOHAMAD OMAR KSSAYRAWI

A thesis submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ARTIFICIAL INTELLIGENCE

2026

Oakland University
Rochester, Michigan

Thesis Advisory Committee:

Dr. Guangzhi Qu, Chair
Dr. Tianle Ma,
Dr. Kaiqi Zhao,

To my father, my mother and Lebanon

ACKNOWLEDGMENTS

I would like to note that without the guidance, encouragement and support of my advisor, Dr. Guangzhi Qu, I would not have made it through my graduate studies. The tips he has given me and the dedication he has showed to the research have been invaluable to this work and I am grateful that I am able to conduct my research under his supervision.

My other thanks go to the members of my thesis committee, Dr. Tianle Ma and Dr. Kaiqi Zhao, who made the time to be on my thesis committee and consider this work.

I would like to acknowledge the assistance and encouragement of graduate students as provided by the Dean of School of Engineering and Computer science of Oakland University, Dr. Louay Chamra.

I am grateful to the Department of Computer Science and Engineering at Oakland University for creating a supportive academic environment and for the resources that made this research possible. I also thank the International Students and Scholars Office for their help in making my experience of living and studying abroad smoother and more manageable.

I am very thankful that the Fulbright Program has helped me to study my graduate degree in the US. My experience as a Fulbright scholar has been a valuable and transformative experience, providing academic, professional and cultural development.

I also thank the people I met during my time at Oakland University for their kindness, encouragement, and friendship, which eased the challenges of living far from home. Their support fostered a sense of community throughout my time abroad.

I am sincerely grateful to my family for their unwavering encouragement and belief in me, and to my friends in Lebanon, whose support and connection have remained strong despite the distance. I especially wish to remember my father, whose values and encouragement continue to inspire my pursuit of education and knowledge.

Lastly, I want to appreciate my personal perseverance and commitment during this process. This work required a lot of strength, interest, and dedication. I am proud of the effort and enthusiasm I put into this research.

This thesis is not just an academic milestone; it also reflects my experience of learning, growing, and contributing to research while studying abroad as part of the global academic community.

Fatima Mohamad Omar Kssayrawi

PREFACE

This thesis was undertaken in partial completion of Master of Science degree in Artificial Intelligence at Oakland University. It is concerned with enhancing the accuracy of retinal vessel graph representations obtained using medical images by trying to overcome the structural inaccuracies that may arise as graphs are built using vessel segmentation.

Retinal vessel graphs are valuable for vascular analysis, but extracted graphs often contain topological inconsistencies such as broken vessels, incorrect connections, and ambiguous junctions. These errors can reduce the reliability of graph-based analysis and limit downstream clinical applications.

To address this problem, this thesis suggests a graph neural network-based model of detecting and fixing structural errors in retinal vessel graphs. To simulate typical topological errors, controlled perturbations are introduced and allow a systematic test of the suggested approach. The idea is to enhance robustness and reliability of the vessel graph representations in biomedical image analysis.

This research was carried out during my graduate studies at Oakland University under the guidance of Dr. Guangzhi Qu. Through this work, I developed a strong interest in research and innovation in artificial intelligence, and I anticipate that I will further expand on this experience with future academic opportunities, such as pursuing a doctoral degree.

ABSTRACT

TOPOLOGY-AWARE CORRECTION OF RETINAL VESSEL GRAPHS USING GRAPH NEURAL NETWORKS

by

Fatima Mohamad Omar Kssayrawi

Adviser: Dr. Guangzhi Qu

Vessel graphs are commonly used to study the structure of retinal vessels, but graphs obtained from vessel segmentation often contain topology errors, including broken vessels and spurious connections. These errors arise from limitations in pixel-level processing and negatively impact graph-based analysis.

Our work addresses this problem by formulating topology correction as a graph-level learning task. We represent connectivity errors as missing or incorrect edges and correct them through a candidate edge prediction task. A perturbation framework is introduced to simulate realistic structural errors, enabling supervised learning across various scenarios.

A graph neural network is used to encode both the geometry and the graph structure of the vessels, enabling the prediction of valid vessel connections. Experimental results show that this approach improves the structural consistency of vessel graphs and enhances the accuracy of connectivity modeling.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

CNN	Convolutional Neural Network
GNN	Graph Neural Network
GCN	Graph Convolutional Network
GAT	Graph Attention Network
GraphSAGE	Graph Sample and Aggregate
VGAE	Variational Graph Autoencoder
MLP	Multi-Layer Perceptron
DR	Diabetic Retinopathy
FOV	Field of View
DRIVE	Digital Retinal Images for Vessel Extraction
TP	True Positive
TN	True Negative
FP	False Positive
FN	False Negative
ROC	Receiver Operating Characteristic
AUC	Area Under the Curve
BCE	Binary Cross-Entropy

CHAPTER ONE
INTRODUCTION

1.1. Background on Retinal Vessel Analysis

Retinal imaging provides detailed information about vascular structure and is widely used for analyzing diseases such as diabetic retinopathy and cardiovascular conditions. The retinal vasculature forms a complex network of blood vessels whose structure reflects both ocular and systemic vascular conditions. Retinal vessel morphology analysis has therefore become a significant area of research in medical image analysis and computer-aided diagnosis systems [1]. Recent developments in deep learning have made a great contribution to automated retinal vessel segmentation. Convolutional neural networks (CNNs) can locate vessels in fundus images and produce binary vessel masks that reflect the location of vessels at the pixel scale [2], [3].

However, segmentation does not explicitly describe the structure of the vascular network. Numerous clinical and computational studies need to know about the connectivity of vessels, branching structures, and junction structures. Segmented vessels are often transformed into graph representations to perform this type of analysis [4]. Graph representations provide a structured model of the vascular network; however, it remains difficult to construct reliable vessel graphs. The topological inconsistencies in the extracted graph can be caused by errors inserted during segmentation or skeletonization, which can produce incorrect vessel connections or missing vessel segments.

1.2. Importance of Vascular Topology in Retinal Analysis

The vascular topology of the retina provides diagnostic information that depends on accurate vessel connectivity, branching, and junction structures [5]. The vascular network of the retina can be analyzed more effectively using graph theory to model these structural properties. In a graphical view, this corresponds to nodes such as endpoints, bifurcations, and junctions, and edges representing segments between nodes. This supports disease detection and tracking through topology-based analysis and the extraction of quantitative biomarkers. However, such analysis is useful only when the extracted graph is accurate. Otherwise, mistakes made during graph construction would affect vessel connectivity.

1.3. Challenges in Vessel Topology Extraction

Although the quality of the results obtained by retinal vessel segmentation is high, it has proved challenging to transform the segmented images into appropriate vessel graphs. As introduced earlier, graph extraction involves multiple processing steps that may introduce structural artifacts. Junction identification errors, connection loss, and redundant branching are common topological errors. For example, crossing vessels can be mistakenly identified as junctions, and a break between segments may split a single vessel into two or more parts. These artifacts lead to inaccuracies in vascular analysis.

Another limitation is the lack of datasets annotated with specific topological errors in retinal vessel graphs. The existing datasets are not annotated with structural connectivity problems of any extracted vessel graph. Developing suitable topology correction techniques becomes more difficult in the absence of such annotations. This highlights the need for datasets where structural errors are explicitly annotated.

1.4. Limitations of Current Segmentation and Skeletonization Methods

Accurate segmentation plays an important role in retinal blood vessel analysis, where problems associated with segmentation occur mostly during skeletonization and graph extraction procedures. Slight segmentation errors may lead to incorrect graph connections. Handling segmentation errors in current methods often relies on geometric rules; however, such methods generally fail to handle the complicated cases found in retinal vessels [6].

1.5. Motivation for Topology-Aware Graph-Based Methods

Representing retinal vessels as graphs enables explicit modeling of vessel connectivity, which is essential for identifying and correcting topology errors introduced during graph extraction. The need for topology-aware techniques that preserve vessel connectivity during graph formation and analysis has been emphasized in recent research [7]. However, existing approaches based on geometric rules often fail in ambiguous configurations, such as crossing or closely spaced vessels, where local features are insufficient to determine true connectivity. Since graph neural networks (GNNs) process graph-structured data and enable information exchange between related nodes, they can learn both geometric and topological patterns and detect topology-related errors [8], [9].

1.6. Overview of the Proposed GNN-Based Approach

We propose a topology-aware system that improves retinal vessel graph generation using graph neural networks. The pipeline begins with the construction of vessel graphs from retinal images through vessel segmentation, skeletonization, and graph extraction.

To generate labeled training data for topology correction, a synthetic perturbation framework is introduced to emulate typical structural errors that may occur during graph

extraction. These perturbations generate graphs containing both valid and invalid connections, supporting supervised learning for topology correction.

The topology correction problem is formulated as a candidate edge prediction problem, where the objective is to determine whether an edge between two nodes is valid and should be present in the corrected vessel graph. We adopt this formulation because it directly models vessel connectivity as an edge prediction problem and enables learning topology-aware patterns beyond local geometric rules, particularly in ambiguous cases such as kissing vessels. Graph neural networks are trained to learn patterns of valid vessel connectivity using node and geometric edge features.

To assess the proposed approach, a geometric heuristic baseline is used for comparison with graph neural network models.

1.7. Research Questions

Given the challenges outlined previously, this work addresses the following research questions:

1. *RQ1:* How can structural topology errors in retinal vessel graphs be systematically modeled and generated to support supervised machine learning training?
2. *RQ2:* How can graph neural networks learn to predict valid vessel connections in perturbed retinal vessel graphs?
3. *RQ3:* How does a topology-aware graph neural network approach compare with geometric heuristic methods for vessel topology correction?

4. *RQ4:* How does model performance vary across different types of vessel topology perturbations, including crossings, kissing vessels, dangling branches, and broken vessels?

1.8. Research Objectives

The objectives of this research are:

1. To build vessel graphs from retinal images using segmentation, skeletonization, and graph extraction algorithms as part of the vascular graph generation pipeline.
2. To develop a synthetic perturbation framework for modeling typical topological errors in vessel graphs.
3. To formulate vessel topology correction as a candidate edge prediction problem.
4. To train graph neural network models that detect valid vessel connections in perturbed graphs.
5. To apply a geometric heuristic baseline for comparison.
6. To test the proposed methods under various topological perturbation scenarios.

1.9. Thesis Organization

- Chapter 2 reviews existing work on retinal vessel segmentation, vessel graph representation, topology analysis, and graph neural networks.
- Chapter 3 formulates the vessel topology correction problem.

- Chapter 4 describes the datasets used in this study and the process of constructing vessel graphs.
- Chapter 5 presents the synthetic perturbation framework.
- Chapter 6 describes the graph neural network models and baseline methods.
- Chapter 7 explains the experimental setup.
- Chapter 8 presents the experimental results and analysis.
- Chapter 9 discusses the findings and limitations of the proposed approach.
- Chapter 10 concludes the thesis and outlines directions for future work.

CHAPTER TWO
RELATED WORK

Retinal vessel topology correction lies at the intersection of vessel segmentation, graph construction, and graph-based learning. While these areas have achieved strong performance individually, they do not fully address the problem of preserving global vascular connectivity. Most existing methods focus on improving pixel-level accuracy or detecting local structures, but errors introduced at early stages often propagate and manifest as incorrect topology in the final vessel graph [13], [14], [24].

A typical retinal image analysis pipeline consists of three stages: vessel segmentation, skeletonization with graph extraction, and downstream analysis such as junction analysis or disease prediction [14], [34], [35], [41]–[44]. Although segmentation models can achieve high accuracy, even minor pixel-level artifacts can lead to significant structural errors after graph construction. For instance, small discontinuities may break vessel continuity, while slight overlaps can create false junctions. These issues directly affect connectivity-based analysis, even when standard segmentation metrics such as Dice or AUC indicate strong performance [13], [24].

Recent work has attempted to incorporate structural priors or connectivity awareness into segmentation models. However, these approaches remain fundamentally limited because they operate at the pixel level and cannot enforce consistency in the final graph representation [22]–[24], [52], [59]. At the same time, graph-based learning methods typically assume that the input graph is already correct, leaving topology errors unaddressed [41]–[44], [47]–[50]. As a result, correcting vessel connectivity at the graph level remains an underexplored problem.

2.1. Retinal Vessel Segmentation

Retinal vessel segmentation extracts vessel structures from fundus images and serves as the foundation for subsequent analysis [13], [14]. Traditional methods using image processing methods, such as vesselness filtering and matched filtering, are vulnerable to noise and may not perform well in poorly contrasted areas or thin and discontinuous vessels [15]–[17].

Recent deep learning approaches, such as U-Net and its variants have greatly enhanced segmentation accuracy by learning both local and global features [19]–[21]. While these methods enhance segmentation accuracy, they do not ensure topological correctness. Pixel-wise predictions are local and can result in small gaps, noise, or over-segmentation, leading to broken vessel structures [52], [54].

This limitation arises because segmentation models optimize for pixel-wise similarity rather than structural correctness. As a result, they cannot ensure that vessels remain connected in a way that reflects true anatomical structure. Even when connectivity-aware losses or attention mechanisms are introduced, the learning process remains constrained to the image domain [22]–[24]. Consequently, errors introduced during segmentation persist and become more pronounced during graph construction [52], [54].

These limitations motivate the need for approaches that operate beyond the pixel level. Rather than attempting to enforce topology indirectly during segmentation, our work addresses connectivity errors directly in the graph representation, where structural relationships can be explicitly modeled.

2.2. Retinal Vascular Junction Detection

Junction detection focuses on identifying bifurcations and crossings, which are critical for understanding vascular structure and supporting tasks such as vessel tracking and classification [25], [32]. Recent deep learning approaches, including heatmap-based localization and graph-based classifiers, have improved the accuracy of detecting these key points [25]–[31].

However, accurate junction detection does not guarantee correct connectivity. Junction detection methods are inherently local: they identify whether a point corresponds to a bifurcation or crossing but do not model how vessel segments are connected across the graph. This creates a disconnect between local detection and global structure. In practice, mistakes are made when segmentation introduces distortions to the vessel structure [52], [54]. For instance, vessels that are close together may artificially merge, even in the absence of a junction. Alternatively, actual connections may be lost due to gaps in segmentations.

Junction detection cannot address these inconsistencies as it does not guarantee global connectivity [30], [31]. This motivates the need for methods that explicitly model vessel segment connectivity. Unlike isolated junction detection, our method uses connectivity as a prediction task over all possible edges, which allows for the removal of incorrect connections as well as insertion of missing connections.

2.3. Graph-Based Representations of Retinal Vasculature

The retinal vascular network can be represented as a graph, which captures both the geometry and connectivity [34], [51]. The graph nodes represent vessel endpoints or junctions, and edges represent vessel segments. These graphs are frequently applied in artery/vein classification, vessel analysis and disease diagnosis [34]-[44].

Despite their advantages, graph representations depend heavily on the accuracy of the graph construction process. This process typically involves skeletonization, node detection, and edge construction, all of which are sensitive to segmentation errors. As discussed earlier, errors introduced during earlier processing stages can lead to structural inconsistencies in the graph. [51], [54].

A key limitation of existing work is that these errors are rarely addressed after graph construction. Most methods treat the graph as a fixed input and proceed with analysis or learning under the assumption that it is correct [34], [41]–[44]. This assumption is problematic because structural errors directly affect the reliability of downstream tasks. For example, incorrect connectivity can lead to misleading conclusions in vessel analysis or disease prediction [34], [35].

These observations suggest that graph construction should not be treated as a final step, but rather as an intermediate representation that may require correction. Our work builds on this idea by explicitly modeling and correcting errors in the graph structure.

2.4. Graph-Based Retinal Vascular Analysis and Graph Neural Networks

Graph neural networks (GNNs) have become a powerful tool for learning from graph-structured data [45], [46]. In retinal analysis, they have been applied to tasks such as classification, segmentation enhancement, and junction analysis [22], [23], [30], [39], [41], [42], [47]–[50]. GCN, GraphSAGE, GAT and VGAE allow information to be shared among connected nodes, so that the model can learn local and global features [45], [46]. GNNs are powerful in learning representations over the graph, but in retinal analysis, GNNs usually assume that the graph is given and is correct [41]-[44], [47]-[50]. This assumption limits their ability to handle real-world data, where graphs often contain missing or incorrect edges due to earlier processing stages [51], [54]. The problem is that most GNN-based methods assume a given graph structure and do not alter the graph. This means they cannot correct errors like dangling edges or invalid edges.

These errors are essentially learned as part of the representation, which may result in poor performance. Our work addresses this limitation by shifting the role of the GNN from representation learning to structure correction. Specifically, we formulate topology correction as an edge prediction problem because connectivity errors naturally correspond to missing or incorrect edges in the graph.

2.5. Limitations of Existing Approaches and Research gap

The previous sections reveal a consistent limitation across existing methods: improvements in segmentation, junction detection, and graph-based learning primarily target local accuracy but do not guarantee global connectivity [22]–[24], [25]–[31], [34], [41]–[44]. As

discussed throughout this chapter, errors introduced during earlier stages propagate through the pipeline and remain unresolved in the final graph representation [51], [54].

These limitations prevent existing methods from addressing topology errors in a direct and systematic way [51], [54]. Rather than implicitly modeling topology errors as a byproduct of segmentation, our approach explicitly models these errors at the graph level. Specifically, we formulate topology correction as a candidate edge prediction problem, in which the model predicts whether two nodes should be connected in the corrected vessel graph.

This formulation is based on the observation that connectivity errors take the form of either missing edges or invalid connections [51], [54]. This allows the model to learn which pairs of nodes should be connected—and which should not—based on both geometric and graph-structural properties.

To enable this learning, common graph-construction errors are modeled using perturbation scenarios, such as crossing vessels, kissing vessels, dangling spurs, and broken vessels [52], [54], [57]. These scenarios provide supervised training data covering a wide range of realistic structural errors.

Unlike pixel-level approaches, this method operates directly on the graph representation, enabling the model to reason about long-range relationships and resolve ambiguities that cannot be addressed locally. As a result, the proposed approach provides a more principled solution to topology correction by explicitly modeling and correcting connectivity errors in retinal vessel graphs.

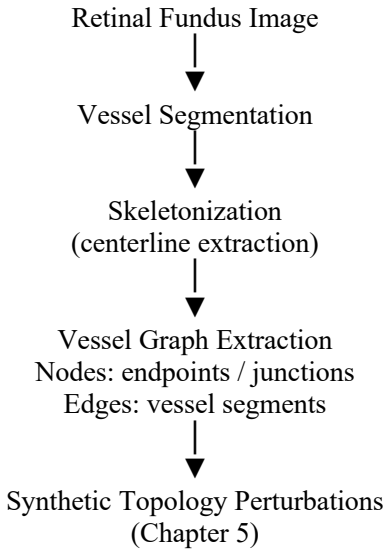
CHAPTER THREE
PROBLEM FORMULATION

3.1. Overview

Retinal vascular analysis increasingly relies on structural representations of the vascular network rather than pixel-level descriptions. As introduced earlier, vessels are modeled as graph. This abstraction (based on graphs) allows topology-aware analysis of vascular connectivity, which is critical to tasks such as artery-vein classification, vessel tree analysis, and disease biomarker extraction [63], [64].

However, vessel graphs of segmented images can be affected by skeletonization and segmentation errors. These artifacts can lead to missing edges, spurious connections, or artificial branches in the graph [65], [66]. Since most downstream analyses involve appropriately connected graphs, these errors may decrease the accuracy of downstream analyses.

This thesis therefore addresses retinal vessel graph topology correction by explicitly formulating it as a graph learning problem to identify and repair incorrect connections.



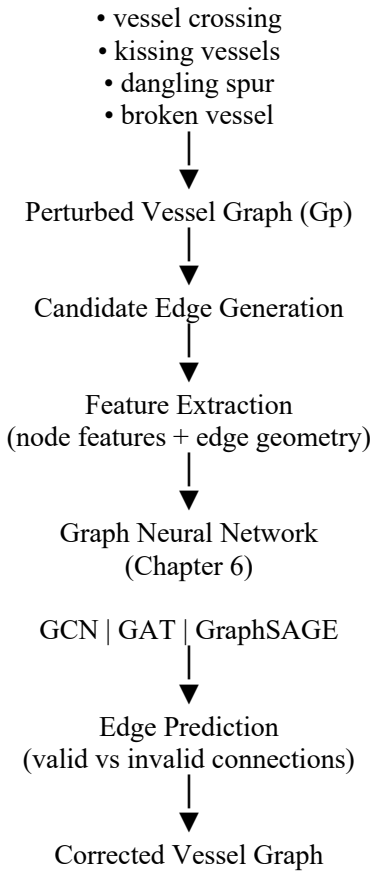


Figure [1]: Pipeline of the proposed retinal vessel topology correction framework. Vessel graphs extracted from segmented images are perturbed to simulate common topology errors. Candidate connections are evaluated using graph neural networks to predict valid vessel connectivity

Each edge $e_{ij} \in E$ represents a vessel segment connecting nodes v_i and v_j. Edge attributes include geometric descriptors such as Euclidean distance, vessel path length, orientation, and tortuosity. These features are a geometric characteristic of segments of the vessels and give useful information to learn connectivity relationships.

This graphical model not only captures the geometry of the retinal vascular network, but also its topology, allowing graph reasoning about its connectivity [63].

3.3. Perturbed Vessel Graphs

As discussed earlier, vessel graphs may contain topology errors introduced during earlier processing stages. To explicitly model these errors, we define a perturbed vessel graph $G_p = (V_p, E_p)$, which represents a graph containing structural inconsistencies.

These inconsistencies directly affect connectivity and impact topology-dependent analysis. In this work, four common perturbation cases are considered:

1. Vessel Crossing: Two vessels can cross in the two-dimensional retinal image plane but be in different vascular branches. Graph extraction algorithms can falsely combine such crossings into one junction, with the result that the vessel graph contains false connectivity.
2. Kissing Vessels: Two vessels can be seen to kiss together by projection effects or segmentation artifacts. Such cases may give false graph edges. There are two variations that are taken into consideration:
    - T-shaped touching vessels
    - Parallel touching vessels.
3. Dangling Spur: Skeletonization artifacts can give rise to short artificial branches, which are not related to actual vessels. Such spurious branches provide false endpoints and unneeded edges to the graph.
4. Broken Vessel: Segmentation discontinuities may leave gaps between segments of a vessel, making one vessel look like two discontinuous pieces in the graph. This leads to the loss of edges which are to be present in the correct topology [65].

3.4. Candidate Edge Generation

Given a perturbed vessel graph G_p, topology correction is formulated by generating candidate edges between nearby nodes.

Let $C = \{(\, v_i, v_j \,)\}$ represent the collection of candidate pairs of nodes that may potentially be vessel connection pairs. Geometric constraints are used to create candidate edges: Spatial closeness amongst nodes, Orientation compatibility, Vessel segment continuity

A feature vector of geometric descriptors of the local vessel structure is created on each candidate pair (vi, vj). The goal is to decide whether a given candidate edge is in the corrected vascular graph as a valid vessel connection.

3.5. Graph Neural Network Formulation

The topology correction problem is formulated as a binary edge prediction task. Given a perturbed graph $G_p = (V_p, E_p)$, a candidate edge set C. The goal is to learn a function f $f(v_i, v_j, G_p) \rightarrow$, where $y_{ij} \in \{0,1\}$ indicating whether the candidate edge between the nodes v I and v j should be present in the repaired graph. The representations of vessel nodes are learned through graph neural networks (GNNs), based on the aggregation of information of their neighbors. By repeating the message passing, every embedding node captures local geometric information and structural context in the vessel graph [67]. This task is explored using several graph neural network architectures, such as Graph Convolutional Networks (GCN) [68], Graph Attention Networks (GAT) [69] and GraphSAGE [70]. These models enable the network to utilize both node attributes and graph connectivity in making predictions of valid vessel connections.

3.6. Learning Objective

The model is trained using supervised learning on perturbed vessel graphs containing labeled candidate edges. For each candidate edge (v_i, v_j), the ground-truth label is defined as

$$y_{ij} = \begin{cases} 1 & \text{if the connection is valid} \\ 0 & \text{otherwise} \end{cases}$$

The model outputs a predicted probability $\hat{y}_{ij}$ representing the likelihood that the candidate edge should exist. Training is performed using binary cross-entropy loss:

$L = -\sum_{(i,j)\in C} \left[y_{ij} \log(\hat{y}_{ij}) + (1 - y_{ij}) \log(1 - \hat{y}_{ij}) \right]$ which encourages the model to distinguish valid vessel connections from invalid ones.

CHAPTER FOUR
DATASET AND GRAPH CONSTRUCTION

4.1. Retinal Image Datasets

This study uses the DRIVE dataset to construct vessel graphs and train the topology correction model. The primary dataset used is DRIVE, which contains 40 color fundus images with expert-labeled vessel masks.

DRIVE has 20 training images and 20 test images. This thesis involves the use of the 20 training images to build vessel graphs and create the dataset to train the topology correction model.

Instead of pixel-level segmentation evaluation, the vessel masks in this work are used to construct structural graph representations of the retinal vasculature by skeletonizing and extracting the graph. The training, validation and testing splits are based on the DRIVE dataset to develop and evaluate the model.

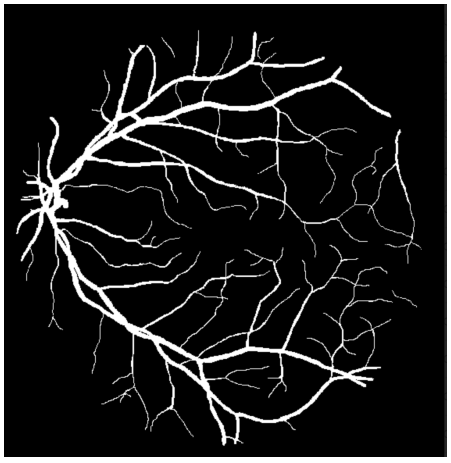


Figure [2]: Binary vessel segmentation mask for the DRIVE image 21_manual1. The white pixels represent the segmented retinal blood vessels, which are later skeletonized to extract the vessel centerline structure used for graph construction.

4.2. Vessel Segmentation Data

The proposed approach uses vessel segmentation masks to construct graph representations of the vascular network. In this study, the input consists of manually annotated vessel masks from the DRIVE dataset. As vessels are represented as thick areas, the masks are skeletonized to create one-pixel-wide centerlines, out of which graph nodes and edges are obtained. The segmentation artifacts may, however, introduce topological errors, such as spurious junctions or isolated vessels, and topology-aware correction algorithms must be developed.

4.3. Skeletonization and Initial Graph Construction

Once the binary vessel segmentation masks have been obtained, the next step would be to transform the segmented vessel regions into a graph representation that can be analyzed using topological methods. Since the segmentation masks mark out vessels as thick foreground regions (as opposed to explicit centerlines), a series of structural processing steps are necessary to extract graph nodes, graph edges, and edge-level geometric features.

4.3.1. Preprocessing of Vessel Masks

The manually annotated vessel mask is loaded as a binary image for each retinal image. Another field-of-view (FOV) mask of the DRIVE dataset exists. This mask removes pixels outside the valid retinal field and therefore the graph building is limited to the visible retinal field. Mathematically: Since M is binary mask of vessel and F is binary mask of FOV, the effective vessel mask is computed as the element-wise product of M and F: $M_{\text{fov}} = M \wedge F$ where pixels are not within the retina field of view, the pixels are zeroed. This will prevent

interruption of the background artifacts along the edges of the picture to be interpreted as vessel structures.

Small connected components (e.g., fewer than 16 pixels) are removed to reduce noise and prevent the creation of spurious nodes and edges. The pixel images pipeline uses images with less than 16 pixels. This process helps reduce the influence of individual noise components or small disguise artifacts in other areas. These artifacts can otherwise introduce unwanted endpoints or small random edges to the output vessel graph.

4.3.2 Skeletonization

After preprocessing, the vessel mask is skeletonized to obtain a one-pixel-wide centerline representation. Let M_c denote the cleaned binary vessel mask. Skeletonization is a way to obtain a binary centerline image S, so that S = skeletonize(M_c) and every vessel is hopefully represented by a connected chain of one-pixel-wide skeleton points. This is needed since the relationship of the centerlines should be evident when developing the graph. The original vessel mask points to the region of the vessel but does not literally point to the segments of vessels initiation, termination or branching points. The skeleton gives a compact structural representation based on which the endpoints, junctions, and pathways of vessels can be defined.

4.3.3 Node Identification from Skeleton Connectivity

After the computation of the skeleton, graph nodes are found by examining the local connectivity of skeleton pixels. The method implemented relies on the 8-neighborhood of each skeleton pixel to identify the number of connected skeleton pixels. The degree of each skeleton pixel p is defined as the number of active neighbors in the 8-neighborhood of p.

According to this degree, there are two types of nodes that are extracted: Endpoint nodes: skeleton pixels of degree 1. Junction nodes: skeleton pixels that have degree at least 3. Degree 2 pixels are represented as interior points on vessel segments and are not represented as graph nodes. The detected nodes are given a node identifier and stored with the following attributes: node ID, spatial coordinates (x, y), node degree, endpoint indicator. So, the initial node set is given as

$$V = \{v_1, v_2, \dots, v_N\}$$

with each node being an endpoint or a junction in the vascular network derived from the skeleton. This model is a faithful representation of the proposed graph abstraction: vascular landmarks are represented as nodes, and edges are continuous segments of vessels between landmarks.

4.3.4 Edge Tracing Along Skeleton Paths

Once the nodes are identified, graph edges are constructed by tracing skeleton paths between pairs of nodes. Tracing continues along valid paths until another node or a dead end is reached. Each edge is represented not only by its endpoints but also by the ordered sequence of skeleton pixels along the path. In addition to edge endpoints, the full pixel polyline is stored in a JSON file for visualization, auditing, and perturbation analysis.

4.3.5 Edge-Level Geometric Feature Extraction

Once the skeleton paths are followed, the geometric properties are computed on an edge. These features describe the geometry and spatial relationships of vessel segments. With the implemented graph construction pipeline, the following features are calculated:

- Path length in pixels (L): the straight geodesic length of the skeleton path traced, made up of horizontal/vertical steps and diagonal steps.
- Euclidean distance: is the distance that the line will travel between source and destination nodes.
- Tortuosity: the ratio of path length/Euclidean distance.
- Orientation: the way the edge between the start and end nodes in the globe is oriented.

The length of path is calculated as the sum of the distance of the skeleton pixels of the edge that are ordered. Horizontal and vertical movements add a length of 1 pixel and diagonal moves add$\sqrt{2}$. If the start and end node coordinates are (x_i, y_i) and (x_j, y_j), then the Euclidean distance is $d_{ij} = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2}$. Tortuosity is then defined as $\tau_{ij} = \frac{L_{ij}}{d_{ij}}$ where larger values indicate more curved or indirect vessel segments. The edge orientation is computed as the angle of the line connecting the start and end nodes: $\theta_{ij} = \text{atan2}\,(y_j - y_i, x_j - x_i)$. These geometric descriptors are a brief description of vessel shapes and continuity, which allow distinguishing between valid and invalid candidate connections during topology repair.

4.3.6 Graph Artifacts Saved for Each Image

Each retinal image is processed by the pipeline to produce files that detail the extracted vessel graph, as well as topology, geometry, visualization and metadata.

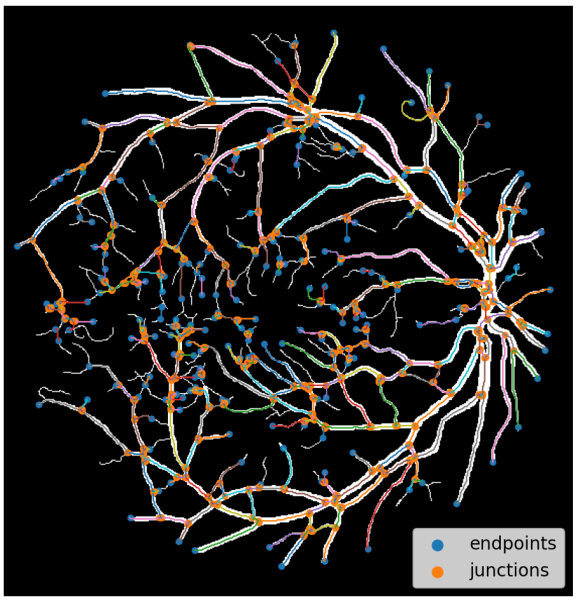


Figure [3]:Visualization of the initial vessel graph extracted from the skeletonized segmentation of the DRIVE image 24_manual1. Vessel centerline paths are displayed as colored polylines, while detected graph nodes are overlaid as points indicating

Figure 3 shows the vessel graph extracted from the skeletonized segmentation of the DRIVE image 24_manual1. The centerlines of vessels are used as the graph edges, and detected nodes are the structural landmark in the network. Instead, the blue markers indicate endpoints (degree one), and the orange markers indicate a location where several vessel segments intersect.

4.3.6.1. Node Representation

The node-level representation of the vessel graph is stored in the file nodes.csv, which contains the list of detected graph nodes and their attributes.

node_id	x	y	degree	is_endpoint
131	185	538	1	1
132	304	561	1	1
133	297	562	1	1
134	282	563	1	1
135	165	64	3	0
136	166	64	3	0
137	165	65	3	0
138	167	65	3	0
139	166	66	3	0
140	318	75	3	0
141	317	76	3	0
142	318	76	3	0
143	333	78	4	0
144	331	79	3	0
145	332	79	4	0

Table [1]: Example rows from nodes.csv generated for the initial vessel graph of the DRIVE image 21_manual1. Each row corresponds to a detected graph node extracted from the vessel skeleton and includes its spatial coordinates, graph degree, and endpoint indicator

Sample rows from nodes.csv are shown in Table 1. The skeleton of the vessel nodes is represented in form of rows, each row contains the spatial coordinates of the node, the degree of the graph and the indicator of the endpoint. The node table has the following fields: node id - a label of a node in the graph; x, y - a position of a node in the image space; degree - the number of edges on a node, is_endpoint - a 0/1 value, which indicates whether a node is a vessel point endpoint or not. Vessel endpoints have a degree of one and a junction structure (bifurcation or candidate crossing) has a degree of more than two. These node properties are used to construct the vessel graph topology upon which one uses.

4.3.6.2. Edge Representation

The edges file of the vessel graph is the connectivity of the nodes, which is stored in the edges.csv file. Table 2 presents some sample rows of edges.csv created on the first vessel graph of the DRIVE image 21_manual1. A row corresponds to a vessel segment joining two nodes and contains some geometric descriptors of the segment. Each edge entry is edge_id - unique identifier of the vessel segment, source node ID, destination node ID,

length-px, Euclidean length, tortuosity, orientation. The tortuosity measure is used to measure the degree to which a vessel section is not straight. A value near one represents relatively straight segments whereas a larger value will represent more curved vessels. The orientation rad value is used to describe the direction of the vessel segment in the image in the global sense. Most vessel segments are found to be slightly tortuous with a tortuosity value of slightly more than one as indicated in Table 2 below which is expected based on the relatively smooth structure of retinal vessels. These geometric descriptors can give useful structural features and are subsequently embedded in the feature representation of the graph learning models.

edge_id	src	dst	length_px	euclidean_px	tortuosity	orientation_rad
0	0	155	62.48528137	60.29925373	1.03625298	1.670464979
1	1	140	40.07106781	38.32753579	1.045490324	1.701623723
2	2	8	18.07106781	16.2788206	1.110096871	1.756144277
3	3	138	18.72792206	17.02938637	1.09974145	2.439335722
4	4	171	75.45584412	68.11754546	1.107729934	1.512040504
5	5	7	14.89949494	13.89244399	1.072489113	1.042721878
6	6	135	6.656854249	6.403124237	1.03962597	0.8960553846
7	9	147	81.55634919	76.65507159	1.063939378	3.010765258
8	10	143	2.414213562	2.236067977	1.079669128	2.034443936
9	11	162	20.55634919	18.35755975	1.119775693	2.083185787
10	12	148	4	4	1	-1.570796327

Table [2]: Example rows from edges.csv generated for the initial vessel graph of the DRIVE image 21_manual1. Each row represents a vessel segment connecting two nodes in the graph and includes geometric features describing the segment length, straight-line distance, tortuosity, and orientation.

4.3.6.3. Pixel-Level Edge Paths

While the edge table describes graph connectivity and geometric features, the exact pixel-level representation of vessel segments is stored in edgepaths.json.

```
[{"edge_id": 0, "src": 0, "dst": 155, "path": [[35, 250], [36, 250], [37,
250], [38, 250], [39, 250], [40, 250], [41, 250], [42, 250], [43, 250],
[44, 250], [45, 250], [46, 250], [47, 250], [48, 250], [49, 250], [50,
249], [51, 249], [52, 249], [53, 249], [54, 249], [55, 249], [56, 249],
[57, 249], [58, 249], [59, 249], [60, 249], [61, 249], [62, 249], [63,
249], [64, 249], [65, 249], [66, 249], [67, 249], [68, 248], [69, 248],
[70, 248], [71, 248], [72, 248], [73, 248], [74, 247], [75, 247], [76,
247], [77, 247], [78, 247], [79, 247], [80, 247], [81, 246], [82, 246],
[83, 245], [84, 245], [85, 244], [86, 244], [87, 244], [88, 244], [89,
244], [90, 244], [91, 244], [92, 244], [93, 244], [94, 244], [95, 244]]},
{"edge_id": 1, "src": 1, "dst": 140, "path": [[37, 323], [38, 322], [39,
322], [40, 322], [41, 322], [42, 322], [43, 322], [44, 322], [45, 322],
[46, 321], [47, 321], [48, 321], [49, 321], [50, 321], [51, 321], [52,
321], [53, 321], [54, 321], [55, 321], [56, 321], [57, 321], [58, 321],
[59, 321], [60, 321], [61, 320], [62, 320], [63, 320], [64, 320], [65,
320], [66, 320], [67, 320], [68, 319], [69, 319], [70, 319], [71, 319],
[72, 319], [73, 318], [74, 318], [75, 318]]}, {"edge_id": 2, "src": 2,
"dst": 8, "path": [[52, 330], [53, 329], [54, 328], [55, 328], [56, 328],
[57, 327], [58, 327], [59, 327], [60, 327], [61, 326], [62, 326], [63,
326], [64, 326], [65, 326], [66, 326], [67, 326], [68, 327]]}, {"edge_id":
```


Figure [4]: Example entries from edgepaths.json generated for the initial vessel graph of the DRIVE image 21_manual1. Each entry stores the pixel-level polyline representing the skeleton path of a vessel segment between two graph nodes.

Each entry includes the following fields: edge_id (edge identifier), src (source node), dst (destination node), and path (a sequence of pixel positions representing the vessel centerline between the two nodes). This representation enables accurate computation of geometric properties such as geodesic length and tortuosity.

4.3.6.4. Graph Metadata

The file metadata.json stores global information about the graph representation. The metadata includes the image ID, image size, feature names, and the number of vessel segments. This file provides a concise summary of the graph structure and feature schema.

The vessel graph is stored using four files: nodes.csv, edges.csv, edgepaths.json, and metadata.json. This representation separates graph structure from the original image while preserving vessel geometry, enabling visualization, inspection, and further processing.

```
{
  "image_id": "21_manual1",
  "image_size": [
    565,
    584
  ],
  "pixel_size_um": null,
  "feature_names": {
    "node": [
      "degree",
      "is_endpoint"
    ],
    "edge": [
      "length_px",
      "euclidean_px",
      "tortuosity",
      "orientation_rad",
      "length_um"
    ]
  },
  "num_edge_paths": 959
}
```


Figure [5]: Example content of metadata.json generated for the initial vessel graph of the DRIVE image 21_manual1. The metadata file stores global information about the image and the feature schema used for nodes and edges in the constructed graph.

4.3.7 Quality Assurance of Graph Construction

After graph construction, quality assurance checks are performed to verify that the extracted graph accurately represents the vessel skeleton. These checks include both structural validation and geometric alignment.

4.3.7.1. Structural Graph Validation

Structural validation ensures that the graph representation is consistent. This includes checking node–edge relationships, identifier validity, absence of self-loops, correct degree values, and basic geometric constraints (e.g., tortuosity ≥ 1 and path length ≥ Euclidean distance).

	base	num_nodes	num_edges	node_ids_contiguous	edges_valid_srcdst	self_loops	degree_mismatches	num_endpoints_csv	num_junctions_csv	tortuosity_<1	length<euclid
0	21_manual1	679	959	True	True	0	0	135	544	0	0
1	22_manual1	805	1078	True	True	0	0	196	609	0	0
2	23_manual1	349	520	True	True	0	0	52	297	0	0
3	24_manual1	1039	1459	True	True	0	0	203	836	0	0
4	25_manual1	762	1041	True	True	0	0	177	585	0	0
5	26_manual1	556	751	True	True	0	0	127	429	0	0
6	27_manual1	918	1332	True	True	0	0	154	764	0	0
7	28_manual1	934	1316	True	True	0	0	189	745	0	0
8	29_manual1	728	1005	True	True	0	0	159	569	0	0

Table [3]: Structural validation statistics for the initial vessel graphs constructed from selected DRIVE images. The table reports the number of nodes and edges, checks for node identifier continuity, validity of edge endpoints, presence of self-loops, and degree consistency between node and edge representations.

The results show that all constructed graphs satisfy these conditions: node identifiers are consistent, edge endpoints reference valid nodes, no self-loops are present, and node degrees are consistent with the edge list. Additionally, geometric constraints are satisfied, confirming the validity of the extracted graph structure.

4.3.7.2. Geometric Alignment with the Vessel Skeleton

Geometric validation compares reconstructed graph paths with the reference vessel skeleton using metrics such as precision, recall, F1-score, IoU, and distance measures.

	base	tp	fp	fn	precision	recall	f1	iou	chamfer_mean	hd95	...	jn_recall	ep_precision	jn_precision	ep_f1	jn_f1
0	21_manual1	6348	3	2	0.999528	0.999685	0.999606	0.999213	0.000426	0.0	...	1.000000	0.992593	1.000000	0.992593	1.000000
1	22_manual1	8024	4	9	0.999502	0.998880	0.999191	0.998382	0.001267	0.0	...	1.000000	0.984694	1.000000	0.984694	1.000000
2	23_manual1	5138	2	5	0.999611	0.999028	0.999319	0.998639	0.000939	0.0	...	1.000000	0.961538	1.000000	0.961538	1.000000
3	24_manual1	8458	0	9	1.000000	0.998937	0.999468	0.998937	0.001042	0.0	...	1.000000	0.980296	1.000000	0.980296	1.000000
4	25_manual1	7053	1	5	0.999858	0.999292	0.999575	0.999150	0.000725	0.0	...	1.000000	0.988701	1.000000	0.988701	1.000000
5	26_manual1	5964	20	32	0.996658	0.994663	0.995659	0.991356	0.007056	0.0	...	0.990698	0.952756	0.993007	0.952756	0.991851
6	27_manual1	7364	1	1	0.999864	0.999864	0.999864	0.999728	0.000136	0.0	...	1.000000	1.000000	1.000000	1.000000	1.000000
7	28_manual1	8023	6	11	0.999253	0.998631	0.998942	0.997886	0.001285	0.0	...	1.000000	0.984127	1.000000	0.984127	1.000000
8	29_manual1	5864	6	8	0.998978	0.998638	0.998808	0.997618	0.001228	0.0	...	1.000000	0.993711	1.000000	0.993711	1.000000

Table [4]: Quantitative agreement between the reconstructed vessel graph paths and the reference vessel skeleton. Metrics include pixel-level precision, recall, F1-score, intersection-over-union (IoU), Chamfer distance, and node-level alignment measures for junctions and endpoints

The results indicate a high level of agreement, with precision above 0.99 and near-perfect detection of junctions and endpoints.

4.4. Corrected Reference Graph Generation

Although the initial graph provides a structural representation, it may contain incorrect connectivity due to projection ambiguities. To address this, a refinement step is applied to correct such errors and produce a reference graph for subsequent perturbation experiments.

4.4.1 Motivation for Crossing Correction

Although the initial graph provides a structural representation, it may contain incorrect connectivity due to projection ambiguities. To address this, a refinement step is applied to correct such errors and produce a reference graph for subsequent perturbation experiments.

4.4.2 Candidate Crossing Nodes

Candidate crossing nodes are identified by selecting nodes with degree four, as such configurations often correspond to projection-induced crossings. For a node $v \in V$ with degree four, the four incident edges are analyzed to evaluate their geometric relationships.

4.4.3 Path Normalization and Arm Representation

Edge paths are represented as ordered pixel sequences and normalized to ensure consistent coordinate representation.

For each candidate node, incident edges are treated as directional arms extending from the node. A local direction vector is estimated using the first few pixels along each arm:

$$u_i = \frac{P_i(k) - P_i(1)}{\| P_i(k) - P_i(1) \|}$$

where k is a small number of points (set to 10). This produces four direction vectors u_1, u_2, u_3, u_4, describing the geometry around the node.

4.4.4 Geometric Pairing of Crossing Arms

To determine whether a node represents a true crossing, the four arms are evaluated based on geometric continuity. The arms can be paired in three possible configurations. For each pairing, the angle between direction vectors is computed.

Valid vessel continuation is characterized by nearly opposite directions (angles close to 180°). The pairing that best satisfies this condition is selected. Additional constraints include:

- Minimum arm length
- Angular threshold (≥ 130°), determined empirically


```
{
  "cross_node": 188,
  "cross_xy": [
    350.0,
    127.0
  ],
  "arms": [
    {
      "edge_id": 199,
      "nbr": 185,
      "length_px": 1.0,
      "width": null,
      "turn_deg": null
    },
    {
      "edge_id": 203,
      "nbr": 187,
      "length_px": 1.0,
      "width": null,
      "turn_deg": null
    },
    {
      "edge_id": 205,
      "nbr": 189,
      "length_px": 1.0,
      "width": null,
      "turn_deg": null
    },
    {
      "edge_id": 206,
      "nbr": 193,
      "length_px": 1.4142135623730951,
      "width": null,
      "turn_deg": null
    }
  ],
```


Figure [6]: Example metadata describing a detected crossing candidate. The record stores the crossing node ID, coordinates, incident arms, geometric pairing, and the resulting graph update applied during correction.

4.4.5 Splitting the Crossing Node

Once a crossing is confirmed, the graph structure is updated. For a node v with neighbors a, b, c, d:

- Remove node v and its incident edges
- Add new edges between paired nodes
- Merge corresponding edge paths
- Recompute geometric features

- Update node degrees

This converts a false crossing into two independent vessel continuations.

183	170	124	3	0
184	171	124	3	0
185	350	126	4	0
186	357	126	3	0
187	349	127	3	0
189	351	127	3	0
190	356	127	4	0
191	357	127	3	0
192	190	128	3	0

Table [5]: Example node entries after crossing correction. Node 188, which originally represented a degree-4 crossing artifact, is removed from the corrected node table.

4.4.6 Updating Edge Features

For each newly created edge, geometric features are recomputed:

$$d_{ab} = \sqrt{(x_b - x_a)^2 + (y_b - y_a)^2}, \tau_{ab} = \frac{L_{ab}}{d_{ab}}$$

where L_{ab} is the geodesic length. Orientation is also recomputed from node coordinates.

4.4.7 Sequential Application Across the Graph

The correction process is applied sequentially to all candidate crossing nodes. Each correction is recorded in metadata for traceability.

Figure [7]: Example record from `crossings_meta.json` showing the geometric pairing of vessel arms at candidate crossing node 188 and the applied graph update in which the node and its four incident edges are removed and replaced with two new edges.

4.4.8 Output of the Corrected Reference Graph

The final corrected graph is stored as:

- nodes_corrected.csv, edges_corrected.csv, edgepaths_corrected.json, crossings_meta.json

These files store both the corrected topology and metadata describing each applied correction.

```
Found 20 images
[EDGE_PATH NORMALIZE] checked=959 swapped=959
[VALIDATE] nodes=653 edges=907 edgepaths=907
[VALIDATE] bad_edges(endpoints missing nodes)=0
[VALIDATE] edgepaths without edges=0
[VALIDATE] edges without edgepaths=0
[ENDPOINT CHECK] mismatched_edges=0 (tol=3.0px)
[Part1] Candidates deg4: 123
[Part1] High-confidence crossings kept: 44
[Part1] Splits applied: 26
[Part1] Saved to: /content/drive/MyDrive/Graph lea
```

Figure [8]: Example execution log of the crossing-correction stage. The algorithm identifies candidate degree-4 nodes, selects high-confidence crossings(44), and applies graph rewiring operations.

CHAPTER FIVE
SYNTHETIC PERTURBATION GENERATION

5.1. Overview of the Perturbation Framework

This chapter presents a framework for simulating topology errors in retinal vessel graphs to generate datasets for evaluating topology correction methods. The perturbations are applied to corrected graphs to avoid introducing unintended artifacts from graph extraction. All perturbations modify the graph locally and are geometrically consistent. In addition to individual perturbations, mixed scenarios combining multiple error types are also generated. The remainder of this chapter describes the perturbation algorithms, dataset generation process, and associated metadata.

5.2. Design Principles of Perturbations

The perturbation framework generates realistic topology errors, including false crossings, kissing vessels, dangling spurs, and broken vessels. These perturbations introduce localized structural changes while preserving the overall vessel geometry. Structural checks are performed after each modification to ensure graph consistency. To ensure reproducibility, perturbations are applied using predefined rules and fixed random seeds.

5.3. Types of Synthetic Perturbations

5.3.1. Case 1: False Crossing

5.3.1.1. Motivation and intuition

Case 1 simulates false vessel crossings, where two vessels that should remain independent are incorrectly connected. These errors commonly arise during graph extraction due to projection effects in 2D retinal images [77]. The corrected graph from Chapter 4 is used as

a reference to identify valid crossing structures, enabling controlled generation of false crossings.

5.3.1.2. Structural definition of the perturbation

In the corrected graph, two independent vessel segments are represented by edges (A,B) and (C,D). To simulate a false crossing, these edges are removed and replaced by a new node J, connected to A, B, C, and D through edges (A,J), (B,J), (C,J), and (D,J). This creates an artificial four-way junction that does not correspond to true vessel connectivity.

5.3.1.3.Importance of this Perturbation

This perturbation models one of the most common topology errors in retinal vessel graphs. It enables the model to distinguish between valid configurations, where vessels cross without connecting, and invalid configurations, where false junctions are introduced.

5.3.1.4. Algorithm for false crossing injection

Perturbations are generated by working with the graph, correcting the crossings (with metadata created in Chapter 4) attached.

Step 1: Load the corrected graph and crossing metadata

The algorithm loads nodes_corrected.csv, edges_corrected.csv, edgepaths_corrected.json, and crossings_meta.json.

Step 2: Identify valid perturbation sites

Valid perturbation sites are identified based on corrected crossing locations.

Step 3: Randomly select crossing sites

A subset of valid sites is selected randomly to generate different perturbed graph variants.

Step 4: Insert an artificial junction node

A new node J is inserted at the crossing location with degree 4.

Step 5: Remove the true vessel continuations

Edges (A,B) and (C,D) are removed from the graph.

Step 6: Add four false crossing edges

Four edges (A,J), (B,J), (C,J), and (D,J) are added to form the false crossing. Geometric features are computed for these new edges.

Step 7: Recompute node degrees

Node degrees and endpoint indicators are updated to maintain consistency.

Step 8: Save perturbation metadata

The algorithm records the inserted node, removed edges, and added edges for traceability.

5.3.1.5. Labels for link prediction using the union graph

A union graph is constructed by combining the corrected and perturbed graphs to support edge prediction.

- Existing edges: present in both graphs (y = 1)
- Removed edges: valid edges removed during perturbation (y = 1)
- Added false edges: invalid edges introduced during perturbation (y = 0)

This formulation enables the model to both remove incorrect edges and restore missing valid connections.

5.3.1.6. Step-by-step example from image 21_manual1

```
Split sites: 26  Valid sites: 26
Chosen sites: 2

======================================================================
[21_manual1] Site #0
  cross_node (removed in Part1): 227   cross_xy: [137.0, 162.0]
  endpoints: A=223, B=229, C=224, D=226
  pseudo-GT pairs: (A,B)=(223,229)  (C,D)=(224,226)
  BEFORE (corrected graph):
    edge AB exists? True  edge_id=963
    edge CD exists? True  edge_id=964
  AFTER (perturbed graph):
    inserted node J=679
    removed true edges: [AB id=963, CD id=964]
    added false edges (to J):
      A–J id=1011,  B–J id=1012,  C–J id=1013,  D–J id=1014
  SANITY CHECKS:
    AB exists after? False  (should be False)
    CD exists after? False  (should be False)
    AJ exists after? True  (should be True)
    BJ exists after? True  (should be True)
    CJ exists after? True  (should be True)
    DJ exists after? True  (should be True)
  DONE

----------------------------------------------------------------------
======================================================================
[21_manual1] Site #1
  cross_node (removed in Part1): 550   cross_xy: [186.0, 426.0]
  endpoints: A=547, B=551, C=548, D=549
  pseudo-GT pairs: (A,B)=(547,551)  (C,D)=(548,549)
  BEFORE (corrected graph):
    edge AB exists? True  edge_id=999
    edge CD exists? True  edge_id=1000
  AFTER (perturbed graph):
    inserted node J=680
    removed true edges: [AB id=999, CD id=1000]
    added false edges (to J):
      A–J id=1015,  B–J id=1016,  C–J id=1017,  D–J id=1018
  SANITY CHECKS:
    AB exists after? False  (should be False)
    CD exists after? False  (should be False)
    AJ exists after? True  (should be True)
    BJ exists after? True  (should be True)
    CJ exists after? True  (should be True)
    DJ exists after? True  (should be True)
  DONE
Saved perturbed graph to /content/drive/MyDrive/Graph learning/DRIVE/dri
```

Figure [9]: Example audit output for Case 1 false crossing perturbations generated from the corrected graph of the DRIVE image 21_manual1. For each selected crossing site, the two true vessel continuations identified during the correction stage (Chapter 4) are removed, and an artificial junction node is inserted at the original crossing location. Four false edges are then created between the endpoints and the inserted node, producing a false four-way crossing configuration used as a negative training example for the link prediction model.

There is an example of the false crossing perturbation on image 21_manual1, which is where the algorithm picks two fixed crossing sites. The audit report is as follows: Split sites: 26 -26 out of the Chapter 4 metadata corrected crossing sites. Valid sites: 26 - There are 26 valid sites to perturb. Chosen sites: 2 -two sites are chosen in this variant of the graph.

Site #0

For the first selected site, the original crossing node removed during Chapter 4 correction is: cross_node = 227**,** cross_xy = [137.0, 162.0]

The four endpoint nodes around the crossing are: $A = 223$, $B = 229$, $C = 224$, $D = 226$

The pseudo-ground-truth vessel pairs inherited from the corrected graph are: $(A, B) = (223{,}229)$, $(C, D) = (224{,}226)$.

Before perturbation, the corrected graph contains the two true edges:

- edge AB: edge_id = 963, edge CD: edge_id = 964

After perturbation:

- a new artificial junction node is inserted: $J = 679$, the two true edges $(223, 229)$ and $(224, 226)$ are removed, four false edges are added:
    - 223-679: edge_id = 1011, 229-679: edge_id = 1012
    - 224-679: edge_id = 1013, 226-679: edge_id = 1014

The sanity checks confirm that the true edges no longer exist after perturbation, while all four false edges do exist.

Site #1

For the second selected site, the original corrected crossing corresponds to:

- cross_node = 550, cross_xy = [186.0, 426.0]

The surrounding endpoints are: $A = 54, B = 551, C = 548, D = 549$

The true vessel pairs are: $(547, 551), (548, 549)$

Before perturbation, the corrected graph contains edge AB: edge_id = 999, edge CD: edge_id = 1000

After perturbation:

- a new junction node is inserted: $J = 680$, the two correct edges are removed, four false edges are added:
    - 547-680: edge_id = 1015, 551-680: edge_id = 1016
    - 548-680: edge_id = 1017, 549-680: edge_id = 1018

Again, the sanity checks verify that the true edges are removed, and the new false junction edges are present.

5.3.1.7. Generated Graph Files After Perturbation

Once the false crossing perturbation has been applied, the resulting graph gets stored with the same format as in Section 4.3.6 and with nodes_perturbed.csv, edges_perturbed.csv, and edgepaths_perturbed.json. A record of perturbations done is stored in labels_case1.json containing the inserted nodes, removed edges, and newly created false edges. In the case of supervised learning a union graph is formed by combining the corrected and perturbed graphs. This combined format and labels of valid connections, missing true connections, or false connections artificially added are recorded in the files nodes_union.csv, edges_union.csv and edgepaths_union.json. Further metadata like perturbation type, seed and image identifier are also logged to enable reproducible experiments.

edge_id	in_ref	in_cur	exists_in_cur	is_added	y_edge_exists	label_reason
1203	1	1	1	0	1	keep_existing
1204	1	1	1	0	1	keep_existing
1205	1	1	1	0	1	keep_existing
1206	1	1	1	0	1	keep_existing
1207	1	0	0	0	1	POS_case1_removed_true_edge
1208	1	0	0	0	1	POS_case1_removed_true_edge
1209	1	1	1	0	1	keep_existing
1210	1	1	1	0	1	keep_existing
1211	1	0	0	0	1	POS_case1_removed_true_edge
1212	1	0	0	0	1	POS_case1_removed_true_edge
1213	0	1	1	1	0	NEG_case1_fake_edge_remove
1214	0	1	1	1	0	NEG_case1_fake_edge_remove
1215	0	1	1	1	0	NEG_case1_fake_edge_remove
1216	0	1	1	1	0	NEG_case1_fake_edge_remove
1217	0	1	1	1	0	NEG_case1_fake_edge_remove
1218	0	1	1	1	0	NEG_case1_fake_edge_remove
1219	0	1	1	1	0	NEG_case1_fake_edge_remove

Figure [10]:Example rows from edges_union.csv generated after applying the Case 1 perturbation. The table indicates whether each edge belongs to the reference graph (in_ref), the perturbed graph (in_cur), and whether the edge should exist in the corrected topology(y_edge_exists). The label_reason field identifies whether the edge corresponds to an unchanged connection, a removed true edge, or an inserted false edge.

edge_id	sample_case	experiment_nam	image_id	run_id	is_perturbation_	edit_type
1198	case1_crossing	seed17	36_manual1	v007	0	normal
1199	case1_crossing	seed17	36_manual1	v007	0	normal
1200	case1_crossing	seed17	36_manual1	v007	0	normal
1201	case1_crossing	seed17	36_manual1	v007	1	case1_crossing
1202	case1_crossing	seed17	36_manual1	v007	1	case1_crossing
1203	case1_crossing	seed17	36_manual1	v007	0	normal
1204	case1_crossing	seed17	36_manual1	v007	0	normal
1205	case1_crossing	seed17	36_manual1	v007	0	normal
1206	case1_crossing	seed17	36_manual1	v007	0	normal
1207	case1_crossing	seed17	36_manual1	v007	1	case1_crossing
1208	case1_crossing	seed17	36_manual1	v007	1	case1_crossing
1209	case1_crossing	seed17	36_manual1	v007	0	normal
1210	case1_crossing	seed17	36_manual1	v007	0	normal
1211	case1_crossing	seed17	36_manual1	v007	1	case1_crossing
1212	case1_crossing	seed17	36_manual1	v007	1	case1_crossing
1213	case1_crossing	seed17	36_manual1	v007	1	case1_crossing
1214	case1_crossing	seed17	36_manual1	v007	1	case1_crossing

Figure [11]:Example rows from the union edge metadata table showing experiment attributes associated with each edge, including the perturbation case (case1_crossing), experiment seed, image identifier, and run identifier.

node_id	x	y	degree	is_endpoint	in_ref	in_cur	is_added	sample_case	experiment_nam	image_id	run_id
808	196	539	3	0	1	1	0	case1_crossing	seed17	36_manual1	v007
809	197	539	3	0	1	1	0	case1_crossing	seed17	36_manual1	v007
810	198	539	3	0	1	1	0	case1_crossing	seed17	36_manual1	v007
811	180	540	3	0	1	1	0	case1_crossing	seed17	36_manual1	v007
812	181	540	3	0	1	1	0	case1_crossing	seed17	36_manual1	v007
813	327	344	4	0	0	1	1	case1_crossing	seed17	36_manual1	v007
814	273	487	4	0	0	1	1	case1_crossing	seed17	36_manual1	v007
815	64	151	4	0	0	1	1	case1_crossing	seed17	36_manual1	v007
816	164	238	4	0	0	1	1	case1_crossing	seed17	36_manual1	v007
817	244	114	4	0	0	1	1	case1_crossing	seed17	36_manual1	v007
818	52	336	4	0	0	1	1	case1_crossing	seed17	36_manual1	v007
819	327	100	4	0	0	1	1	case1_crossing	seed17	36_manual1	v007
820	88	434	4	0	0	1	1	case1_crossing	seed17	36_manual1	v007
821	307	507	4	0	0	1	1	case1_crossing	seed17	36_manual1	v007
822	178	326	4	0	0	1	1	case1_crossing	seed17	36_manual1	v007

Figure [12]: Example rows from nodes_union.csv illustrating the combined node representation. The columns indicate whether each node originates from the corrected graph (in_ref), the perturbed graph (in_cur), or was inserted during the perturbation process (is_added).

5.3.2 Case 2: Kissing Vessels

5.3.2.1.Motivation and intuition

Kissing vessels refer to local contact between two adjacent but unconnected vessel segments. This occurs when nearby vessels are incorrectly linked during skeletonization or graph extraction due to their spatial proximity. In the image, these vessels appear to touch or “kiss,” despite having no true anatomical connection.

Unlike Case 1, which introduces a false junction, Case 2 creates a short artificial bridge between two distinct vessel segments. Two geometric configurations are considered:

- T-shaped kissing: the local angle between vessel directions is close to 90°.

- Parallel kissing: the local angle is close to 0° or 180°.

These configurations represent common errors in graph extraction.

5.3.2.2. Structural definition of the perturbation

Let two distinct edges e_1 and e_2 have polyline paths P_1 and P_2. The perturbation selects two points $p \in P_1$and $q \in P_2$ such that the Euclidean distance between them is small and the edges are not already connected.

If the pair satisfies the geometric conditions for either T-shaped or parallel configurations, a short artificial bridge is introduced between p and q. If either point does not correspond to an existing node, a new node is inserted by splitting the corresponding edge.

This results in a localized spurious connection while preserving the overall graph structure.

5.3.2.3. Importance of this perturbation

This perturbation captures subtle topology errors where spatially close but anatomically unrelated vessels are incorrectly connected. Although these bridges are small, they can significantly affect connectivity analysis, vessel tracing, and structural interpretation. By simulating both T-shaped and parallel configurations, the model is exposed to realistic local connectivity errors.

5.3.2.4. Geometric modes: T-shaped and parallel kissing

Candidate pairs are classified based on the angle between local tangent directions.

5.3.2.4.1.T-shaped kissing

A candidate pair is classified as T-shaped when the local tangent angle is close to 90°, forming a perpendicular configuration. $|\theta - 90^\circ| \leq \theta_T$ where θ_T is the tolerance threshold.

5.3.2.4.2.Parallel kissing

A candidate pair is classified as parallel when the angle is close to 0° or 180°, representing aligned or opposite directions. $\theta \leq \theta_{\text{parallel}} \text{or} \theta \geq 180^\circ - \theta_{\text{parallel}}$ where θ_{parallel} is the tolerance threshold.

5.3.2.5. Algorithm for kissing vessel injection

The perturbation process starts with the corrected graph, and the topological consistency of the base structure is made first, followed by the introduction of the false bridge.

Step 1: Load the corrected graph

The algorithm loads nodes_corrected.csv, edges_corrected.csv, and edgepaths_corrected.json.

Step 2: Mine candidate edge pairs

Edge pairs are selected based on spatial proximity, minimum distance threshold, and angular compatibility with the selected mode (T-shaped or parallel).

Step 3: Select candidate pairs for one variant

A subset of valid candidates is randomly selected. Each region is modified at most once to avoid overlapping perturbations.

Step 4: Determine whether the selected points already correspond to nodes

The closest points p and q are evaluated. If they do not correspond to existing nodes, new nodes are inserted.

Step 5: Split edges if necessary

If a selected point lies within an edge, the edge is split into two segments to maintain valid node connections.

Step 6: Insert the false bridge edge

A new edge is added between the selected nodes. Its geometric features are computed and stored.

Step 7: Recompute node degrees

Node degrees and endpoint indicators are updated to ensure consistency.

Step 8: Save perturbation metadata

The algorithm records selected edge pairs, inserted nodes, split edges, and the added bridge edge in labels_case2.json and related audit files.

5.3.2.6. Labels for link prediction using the union graph

A union graph is constructed by combining the corrected and perturbed graphs.

5.3.2.6.1. Meaning of union nodes

The nodes_union.csv file contains nodes from both graphs with indicators:

- in_ref = 1: node exists in the reference graph
- in_cur = 1: node exists in the perturbed graph

- is_added = 1: node introduced during perturbation

New nodes correspond to inserted bridge endpoints or split locations.

5.3.2.6.2. Meaning of union edges

The edges_union.csv file includes all edges with labels:

- Existing edges: $y = 1$ (valid connections)
- Added bridge edges: $y = 0$ (invalid connections)

This setup enables the model to preserve correct structure while removing artificial connections. Unlike Case 1, this perturbation primarily introduces false edges without removing valid ones.

5.3.2.7. Examples about case 2

5.3.2.7.1. Step-by-step example*:* T-shaped kissing

In the following example (figure 14), the algorithm had the following results: Candidate pairs found = 44, mode = T, pair = ((4,162)) min distance = 1.00 px local angle = 74.7. The closest points chosen are point (p) along edge 4 with index 68 where (p) equals (188,122), and point (q) along edge 162 with index 31 where (q) equals (189,122).

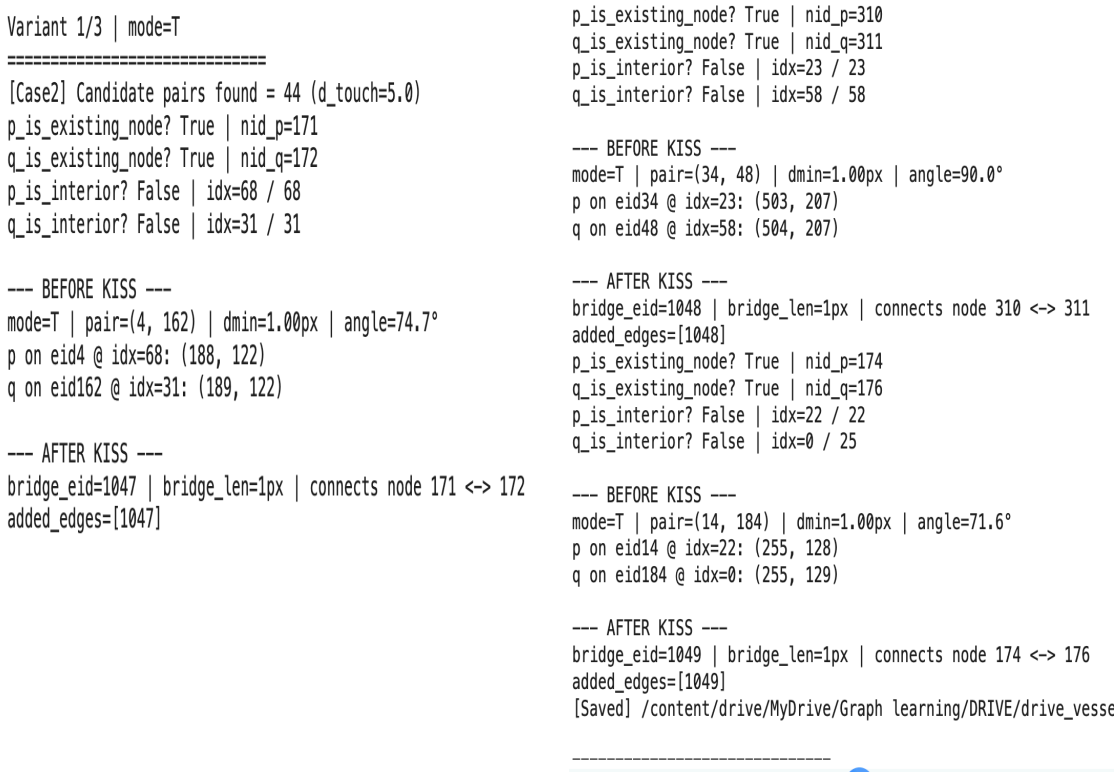


Figure [13]: Console output illustrating three kissing-vessel perturbation edits (Case 2) in T-mode for image 21_manual, where nearby vessel segments are detected and connected by short artificial bridge edges to simulate kissing vessel topology errors.

According to the output of the audit, these two points are already graph nodes: nid_p = 171, nid_q = 172. No splitting is needed since neither of the points is on the interior of an edge. When the perturbation has taken place, there is an inserted bridge edge bridge_eid = 1047, bridge_len = 1 px, between node 171 and node 172. The second T-shaped one indicates pair = (34,48), minimum distance = 1.00 px, angle = 90.0 with existing nodes 310 and 311, linked with bridge_eid =1048. Another T-shaped example reports pair = ((14,184)), minimum distance = 1.00 px, angle = 71.6° with existing nodes 174 and 176, connected by: bridge_eid = 1049. The following are examples of Case 2: two graph nodes already present are physically near but unlinked in the reference graph, and a short false bridge is added between them.

5.3.2.7.2. Step-by-step example: parallel kissing

In the below figure 14 example, the selected pair = (33,834), minimum distance = 1.41 px, angle = 0.0°. The selected points are p = (290,284), q = (291,283).

```
p_is_existing_node? False | nid_p=None
q_is_existing_node? False | nid_q=None
p_is_interior? False | idx=25 / 25
q_is_interior? False | idx=0 / 36

--- BEFORE KISS ---
mode=parallel | pair=(33, 834) | dmin=1.41px | angle=0.0°
p on eid33 @ idx=25: (290, 284)
q on eid834 @ idx=0: (291, 283)

--- AFTER KISS ---
bridge_eid=1444 | bridge_len=1px | connects node 904 <-> 905
added_nodes=[904, 905]
added_edges=[1444]
Nodes near q: []
p_is_existing_node? False | nid_p=None
q_is_existing_node? False | nid_q=None
p_is_interior? False | idx=40 / 40
q_is_interior? False | idx=0 / 59

--- BEFORE KISS ---
mode=parallel | pair=(1330, 1338) | dmin=4.00px | angle=18.4°
p on eid1330 @ idx=40: (456, 246)
q on eid1338 @ idx=0: (456, 250)

--- AFTER KISS ---
bridge_eid=1445 | bridge_len=4px | connects node 906 <-> 907
added_nodes=[906, 907]
added_edges=[1445]
```

Figure [14]: Console output illustrating kissing-vessel perturbations (Case 2) in parallel mode for image 21_manual, where two nearby vessel segments are connected by short artificial bridge edges to simulate kissing vessel topology errors.

The result of the audit is that neither of these points is represented in a graph node:

- p_is_existing_node = False, q_is_existing_node = False

Thus, the algorithm will add two new nodes added_ nodes = [904, 905] and add a bridge edge bridge_eid = 1444, bridge_len = 1 px that will have node 904 add a bridge edge to node 905.

A second parallel example reports pair = (1330,1338), minimum distance = 4.00 px, angle = 18.4°. Once again, both chosen points are not already nodes, and therefore the algorithm adds: added nodes = [906, 907] and bridge_eid = 1445, bridge length = 4 px. As these examples illustrate, the perturbation handles such instances where the bridge must be

inserted between points which are already on vessel paths, but which are not already graph nodes. The two examples illustrate the two prominent geometric shapes of Case 2:

- T-shaped kissing, when the local directions of the vessels are orthogonal.
- Parallel kissing, where the local vessels head directions are facing each other.

In the 2 cases, we have the perturbation establishing a transient artificial contact between two already segments of vessels that are close together. The algorithm may tie the given nodes or introduce new nodes and partition the existing edges, by the local graph structure, to ensure consistency of the graph. The resulting graph thus has a realistic local fake association which can be utilized as negative training example to the link prediction model.

5.3.2.8. Generated graph files after perturbation

After applying the kissing vessel perturbation, the graph is stored in the same format as in the previous section. The resulting files include nodes_perturbed.csv, edges_perturbed.csv, and edgepaths_perturbed.json. Perturbation labels and metadata are stored in labels_case2.json and audit_case2_header.json. A union representation is also generated, consisting of nodes_union.csv, edges_union.csv, and edgepaths_union.json, to support link prediction training. In this representation, newly introduced nodes are identified, and the inserted bridge edge is labeled as a false connection to be removed during correction.

5.3.3 Case 3: Dangling Branch

5.3.3.1. Motivation and intuition

Case 3 simulates a small spurious branch that does not correspond to a true anatomical vessel. These artifacts typically arise from segmentation noise, skeletonization errors, or local tracing inaccuracies. In the graph representation, they appear as short dangling edges attached to otherwise valid vessel segments or junctions.

Unlike Case 1 and Case 2, which introduce false connections between existing vessels, this perturbation creates a new branch extending outward from the vessel structure. The goal is to train the model to detect and remove such local branch-like artifacts while preserving valid connectivity.

5.3.3.2. Structural definition of the perturbation

Let $G = (V, E)$ represent the corrected graph. Case 3 introduces a short spur edge e_{spur} connected to an existing location in the graph.

The perturbation may include:

- a new terminal spur node
- a new spur edge
- a split node (if the attachment occurs on an edge interior)

This spur edge is labeled as a false edge to be removed during topology correction.

Two attachment modes are considered:

- spur on an edge
- spur on a bifurcation

5.3.3.3. Importance of this perturbation

Dangling branches are common local artifacts in skeleton-based vessel graphs. Although small, they affect endpoint detection, branching structure, and connectivity analysis. This perturbation allows the model to distinguish between valid vessel continuations and artificial branches, improving robustness in topology correction.

5.3.3.4. Two modes of spur insertion

Case 3 generates dangling branches in two structurally different ways.

5.3.3.4.1. Spur on an edge

The spur is attached to the interior of an existing edge. Since edge interiors are not graph nodes, the edge is first split at the attachment point. A new node is inserted, and the spur edge is connected to this node.

5.3.3.4.2. Spur on a bifurcation

The spur is attached directly to an existing degree-3 node. No edge splitting is required, and a new edge is added extending from the junction.

5.3.3.4.3. Difference between the two modes

The difference lies in the attachment location:

- On-edge requires splitting an existing edge before attaching the spur
- On-junction attaches directly to an existing node

Both modes create a dangling branch but differ in how the graph is modified locally.

5.3.3.5. Candidate selection for spur perturbation

Candidate locations are selected from the corrected graph.

For spur on an edge:

- edges must be sufficiently long
- attachment points must be away from endpoints

For spur on a bifurcation:

- nodes must have degree 3
- incident edges must be sufficiently long

These constraints ensure that the perturbation does not affect critical regions of the graph.

5.3.3.5.2. Candidates for spur on a bifurcation

A valid candidate junction node is an example of a node whose degree is three and whose incident edges are not so short. This restriction shuns the perturbation instead of centering the sound bifurcation arrangements instead of the local areas of the graphs. The presence of a degree-3 node is explainable by the fact that such a division of the retinal vasculature would be a degree-3 node anyway since any natural division of the retinal vasculature would be a degree-3 node since the branch would naturally be located there where a fake additional branch would be visible.

5.3.3.6. Algorithm for dangling branch injection

The perturbation process is applied to one graph variant at a time, with multiple spur edits inserted per run.

Step 1: Load the corrected graph

The algorithm loads nodes_corrected.csv, edges_corrected.csv, and edgepaths_corrected.json. The archived polylines are then recreated to get the edge attributes so that the working graph with updated geometry and adjacency information can be had.

Step 2: Build candidate lists

Candidate edges and junctions are identified based on structural constraints.

Step 3: Choose the perturbation mode

The algorithm selects either on-edge or on-junction mode (randomly or predefined).

Step 4: Select a host location

A valid edge or junction is selected based on the chosen mode.

Step 5: Determine the attachment point

- On-edge: a point is sampled along the edge
- On-junction: the node itself is used

Step 6: Split the edge if necessary

If the spur is attached to an edge, the edge is split and a new node is inserted.

Step 7: Determine the spur direction

The direction is based on local vessel geometry, with added random variation.

Step 8: Set the spur length and endpoint

A short length is selected, and collision checks ensure the new node does not overlap existing structures.

Step 9: Add the spur node and spur edge

A new node and spur edge are inserted, marked as a false edge.

Step 10: Recalculate graph statistics.

Node degrees, adjacency, and geometric features are updated to maintain consistency.

5.3.3.7. Step-by-step example of Case 3

```
============================
RUN 001 | modes=['on_edge', 'on_junction', 'on_edge']
============================
[GLOBAL USED] edges=0 junctions=0

--- Inject spur 1/3 | mode=on_edge ---

[BEFORE | on_edge]
  selected_edge_id=280  src=235(deg=3)  dst=286(deg=3)
  edge_length_px=57.28
  candidate_reason=ok
  n_safe_points=25  example_attach=(181.0,182.90942169310804)
  insert_new_node_at_s=40.23  attach_point=(183.0, 187.4283869528603)
[AFTER | split]
  new_split_node_id=679 at (183.00,187.43)
  removed_old_edge=280
  new_edges_from_split: e1=1011 (src=235→679), e2=1012 (679→286)

[AFTER | spur added]
  new_spur_node_id=680 at (178.95,187.43)
  new_spur_edge_id=1013  src=679 dst=680
  spur path endpoints: (183.0, 187.4283869528603) -> (178.95442425151822, 187.42
  spur_length_px=4.05  euclidean_px=4.05  tortuosity=1.00
  spur_orientation_rad=-3.141  spur_angle_deg=-179.98
  spur_dir_dxdy=(-1.000,-0.000)  jitter_deg=21.16
  degree(attach_node) AFTER = 3
  degree(spur_end_node) AFTER = 1
```

Figure [15]: Example audit output for a dangling-branch perturbation inserted on an existing edge. The host edge is first split at a safe interior location, after which a short spur edge is attached to the new split node.

In Figure 15-17 there is a run where 3 spur edits are added with the mode sequence as on edge, on junction, on edge.

Example 1: On an edge, spur.

During the first edit, the algorithm picks the selected edge loaded with the id 280, the source node loaded with 235, the degree loaded with 3, the destination node loaded with 286, and the degree loaded with 3, the length of the host edges loaded with 57.28 px.

The audit reports that there are valid candidate and safe interior attachment points. An arc-length position is picked: s = 40.23 that corresponds to the position of attachment (187.43, 183.0). The original edge 280 is then removed and replaced by two new edges: 1011 : 235 → 679, 1012 : 679 → 286 where node 679 is the newly inserted split node. After the split, the spur itself is added: new spur node = 680, new spur edge = 1013, edge 1013 connects 679 → 680. The new spur is of length = 4.05 px, Euclidean distance = 4.05 px, and has a tortuosity = 1.00. This is an artificial branch that is newly created; hence, the spur edge 1013 will subsequently be labeled as negative edge to remove.

Example 2: Spur inserted on a bifurcation

```
--- Inject spur 2/3 | mode=on_junction ---

[BEFORE | on_junction]
  selected_bifurcation_node=162  degree=3
  attach_point=(422.0, 97.0)
  incident_edges=[9, 19, 161]
  incident_lengths_px=[20.56, 93.47, 91.84]

[AFTER | spur added]
  new_spur_node_id=681 at (425.37,94.35)
  new_spur_edge_id=1014  src=162 dst=681
  spur path endpoints: (422.0, 97.0) -> (425.36837248204347, 94.35
  spur_length_px=4.29  euclidean_px=4.29  tortuosity=1.00
  spur_orientation_rad=-0.666  spur_angle_deg=-38.18
  spur_dir_dxdy=(0.786,-0.618)  jitter_deg=2.85
  degree(attach_node) AFTER = 4
  degree(spur_end_node) AFTER = 1
```

Figure [16]: Example audit output for a dangling-branch perturbation inserted on a bifurcation node. In this case, no edge splitting is required because the attachment point already corresponds to an existing degree-3 node.

The second edit of the structure in on-bifurcation mode and selected-bifurcation-node = 162, degree = 3, attachment point = (422.0, 97.0) as the selection, the algorithm will embark on, resultant in the output shown below:

The incident edges are: 9, 19, 161 with lengths approximately: 20.56 px, 93.47 px, 91.84 px. Because the point of attachment is already an endpoint of a graph a splitting of the edges is not needed. The algorithm directly adds a new spur node = 681, new spur edge = 1014, edge 1014 connects 162 → 681. The spur has: length = 4.29 px, tortuosity = 1.00. After a break, the degree of the attachment node, r, is disturbed and becomes 4, and the new spur endpoint has a degree of 1.

Example 3: Another spur inserted on an edge

```
--- Inject spur 3/3 | mode=on_edge ---

[BEFORE | on_edge]
  selected_edge_id=236  src=211(deg=3)  dst=240(deg=4)
  edge_length_px=51.80
  candidate_reason=ok
  n_safe_points=25  example_attach=(159.0,236.88216442042778)
  insert_new_node_at_s=16.72  attach_point=(160.0, 241.05828567430598)
[AFTER | split]
  new_split_node_id=682 at (160.00,241.06)
  removed_old_edge=236
  new_edges_from_split: e1=1015 (src=211→682), e2=1016 (682→240)

[AFTER | spur added]
  new_spur_node_id=683 at (156.84,243.48)
  new_spur_edge_id=1017  src=682 dst=683
  spur path endpoints: (160.0, 241.05828567430598) -> (156.83913151558522, 243.48
  spur_length_px=3.98  euclidean_px=3.98  tortuosity=1.00
  spur_orientation_rad=2.488  spur_angle_deg=142.54
  spur_dir_dxdy=(-0.794,0.608)  jitter_deg=-16.32
  degree(attach_node) AFTER = 3
  degree(spur_end_node) AFTER = 1
removed_edge_ids used for union: [236, 280]

[CHECK] spur edges in union:
     edge_id  is_spur  y_edge_exists          label_reason
909     1013        1              0  NEG_remove_this_edge
910     1014        1              0  NEG_remove_this_edge
913     1017        1              0  NEG_remove_this_edge
```

Figure [17]: Example audit output showing a second edge-based spur insertion and the resulting union-edge labels, where the newly added spur edges are marked as false edges that should be removed.

The third edit depicts the algorithm making a conjecture in again on the on edge, and selecting selected edge id = 236, source node = 211, and degree = 3, destination node = 240, degree = 4 and Host edge length = 51.80 px. The split occurs at: s = 16.72 and it forms a new split node = 682. The original edge 236 is removed and replaced by: 1015 : 211 → 682, 1016 : 682 → 240. The algorithm then inserts a new spur: new spur node = 683, new spur edge = 1017, edge 1017 connects 682 → 683, The spur has: length = 3.98 px ; tortuosity = 1.00. An audit would then make the deleted reference edges in the union

construction explicit: [236, 280] and make certain that the spur edges of the union are: [1013],[1014],[1017] labelled with: y edge exists = 0, label reason = NEG remove this edge.

This is a type that demonstrates the two types of Case 3. In the on_edge mode: A legal edge of a vessel is selected followed by splitting this edge to make a safe interior point and a new split node is formed, and the spur is attached to the new point. In the on-junction mode: the spur will be glued on a degree-3 bifurcation which exists; hence, this will not require edge splitting. The result of the learning is also analogous i.e. the graph currently has a short hanging branch that was not present in the fixed reference topology and thus needs to be removed.

5.3.3.8. Labels for link prediction using the union graph

After generating the perturbed graph, a union graph is constructed by combining the corrected reference graph and the perturbed graph. This representation is used to generate labels for link prediction.

5.3.3.8.1. Meaning of union nodes

The file nodes_union.csv contains nodes from both the corrected and perturbed graphs. A node may appear in the reference graph, the perturbed graph, or both.

The columns are defined as follows:

- in_ref = 1: node exists in the reference graph
- in_cur = 1: node exists in the perturbed graph
- is_added = 1: node introduced during perturbation

In Case 3, newly added nodes include:

- split nodes (created when an edge is divided)
- terminal spur nodes

5.3.3.8.2. Meaning of union edges

The file edges_union.csv contains all edges from both graphs, along with labels indicating whether each edge should exist in the corrected topology.

In Case 3, three types of edges are considered:

1. Untouched valid edges: Present in both graphs → $y = 1$, label_reason = keep_existing
2. Removed reference edges (due to edge splitting): Present in the reference graph but not in the perturbed graph → $y = 1$, label_reason = POS_removed_true_edge
3. New spur edges: Artificial edges introduced during perturbation → $y = 0$, label_reason = NEG_spur_edge

This labeling scheme ensures that the model learns to preserve valid structure, restore removed edges, and eliminate artificial spur edges.

5.3.3.8.3. Importance of Union in Case 3

The union representation is particularly important in Case 3 because the perturbation modifies the graph in two ways:

- introduces a false spur edge

- may remove a valid edge during splitting

Without the union graph, the model would only observe the perturbed graph and would not recognize that certain removed edges belong to the correct topology.

5.3.3.9. Generated graph files after perturbation

After applying the dangling branch perturbation, the graph is stored in the same format as in previous cases. The main files include:

- nodes_perturbed.csv, edges_perturbed.csv, edgepaths_perturbed.json

Perturbation metadata is stored in:

- labels_case3.json, audit_case3.json, audit_case3_summary.csv

The union representation is stored as:

- nodes_union.csv, edges_union.csv, edgepaths_union.json

This structure ensures reproducibility and supports supervised training of graph neural networks.

5.3.4 Case 4: Broken Vessels

5.3.4.1. Motivation and intuition

Case 4 models a scenario where a true vessel segment is broken and one of its endpoints is incorrectly reconnected to a nearby unrelated vessel. This introduces two simultaneous errors: a missing true connection and a false connection.

Unlike simpler cases, this perturbation combines both disconnection and incorrect reconnection, resulting in a structurally inconsistent local topology. The objective is to train the model to restore the correct vessel continuity while removing the erroneous bridge.

5.3.4.2. Structural definition of the perturbation

Let $G = (V, E)$ denote the corrected reference graph. The perturbation proceeds in three stages:

1. A valid edge is selected and split at an interior point, creating two disconnected endpoints.
2. One of the endpoints is selected for incorrect reconnection.
3. A false bridge is added between this endpoint and a nearby vessel segment.

As a result:

- the original vessel continuity is broken
- a false connection is introduced
- additional valid edges are created due to splitting

This makes Case 4 a combination of missing positive edges and incorrect negative edges.

5.3.4.3. "Broken Vessel" Definition

A broken vessel refers to a segment that should be continuous but is interrupted in the graph representation. The original edge is split into two parts, creating two endpoints that should be reconnected.

In this perturbation, one of these endpoints is instead connected to a different vessel, forming a false junction. The model must therefore:

- remove the incorrect bridge
- restore the original connection between the broken endpoints

5.3.4.4. Importance of this perturbation

Case 4 is more complex than previous cases because it combines:

- loss of true connectivity
- creation of false connections
- modification of node degrees
- local structural inconsistency

This makes it particularly valuable for evaluating whether the model can reason beyond local geometry and recover biologically plausible vessel continuity.

5.3.4.5. Candidate edge selection for vessel breaking

Candidate edges are selected based on structural stability to ensure realistic perturbations.

An edge is considered valid if:

- its geodesic length exceeds a minimum threshold
- its endpoint nodes have moderate degree (≤ 3)
- it is not connected to highly complex junctions

- its polyline contains sufficient points for safe splitting

In practice, a minimum length of approximately 40 pixels is used, along with constraints on node degree and path length.

5.3.4.6. Algorithm for broken-vessel generation

For each run, the algorithm starts from the corrected reference graph and applies multiple Case 4 edits sequentially. Each edit follows the same sequence of operations.

Step 1: Select an edge to break

The algorithm then constructs a table of valid edges between candidates. Breached on one of the candidate edges. The findings of the audit are the chosen edge ID had destination nodes, its length and the rationale why it was regarded as viable candidate. The algorithm, e.g. the one in the shown run, is run on: edge_id_to-break = 8, endpoints = (97, 417), length-px = 83.63. This edge is picked because it is long enough and is not linked to extremely complicated crossroads.

Step 2: Break the selected edge

The chosen edge is then cut at a point within it, by removing a small piece of course. This causes a gap and splits the initial edge in two pieces that are separated. The old edge is removed, and two new endpoint nodes are introduced where the break is, and 2 new valid edges are created, one per piece that remains. Under example: the initial edge 85 is removed, new break nodes 679 and 680 are inserted and the environment replaces the initial edge with new ones 1011 and 1012. This is a legitimate amount of vessel that is new. No real error lies in the values of these edges, but that the continuity between the two broken endpoints is no longer present.

Step 3: Choose one broken endpoint to reconnect wrongly

After breaking the edge, the algorithm picks one of the two new break endpoints and uses one of them as the endpoint that will be reconnected improperly. The other is not fettered. This is randomly selected and when it is selected it describes the attached broken endpoint, and the free broken endpoint. The edge between the two removed terminals is the edge which should be taken up again later.

Step 4: Find a nearby target vessel

This algorithm then tries to find an extra, nearby, edge that is irrelevant to the broken vessel. Its target must not surpass a radius of the search, and they must have a safe central location where a new node can be introduced. This will re-relate assumptions in an implausible but flawed way. Information on spatial proximity to the selected broken endpoint is used to determine the target edge. In the sample, the algorithm found that there was a nearby target edge, that the nearest target edge was found at 643 which was the split point chosen, and the distance was approximated at 26.02 pixels.

Step 5: Insert a node on the target vessel

To ensure the broken endpoint is joined to the next ship, a new node is placed at the edge of its destination. This splits up the target edge to two good sub-edges. Subsequently, the target vessel is modified accordingly: the original target edge gets revealed, a new node is added to it, and the first target node is replaced by two new ones which are valid. As illustrated: inserted target node = 681, target edge 643 is split, to form new edges 1013 and 1014. Before the introduction of the dropped bridge, this target node-added node is a degree 2 node in the sense that it simply is inserted in the middle of a legal vessel segment.

Step 6: Add the wrong bridge edge

A fake edge between the selected broken node and the target node inserted is then added. This results in the wrong reconnection. Wrong bridge edge in following example = 1015 with using node 680 and 681. The inserted node and bridge mean the inserted node will be a target degree of 3 which is like a false junction like formation.

This is the undesired critical point in Case 4: here the critical point occurs in the perturbed graph but should not be occurring in the fixed topology.

Step 7: Update graph structure and features

Any newly added and modified edges are then updated by identifying their geometric features (geodesic length then Euclidean distance and tortuosity, orientation, degrees of Nodes and endpoint markers are also re-calculated) to make the graph consistent with the supposedly new and changed connection.

5.3.4.7. Step-by-step interpretation of the example

An example of the perturbation of case 4 happens is shown in the below figure 18. It begins with a legal vessel edge, edge 85, between nodes 97 and 417. This perturbation-structurally appropriate edge is long. It is splintered in the middle by the algorithm to form two new endpoint nodes, 679 and 680, and has created two valid partial edges between the original vessel edge, 1011 and 1012. At this point the ship has been shattered. The real continuity that is to be is between the two newly created break ends. To say it differently, the lost real edge would be the one that would bring back the broken ship. Then, the algorithm finds a nearby unrelated vessel and finds edge target 643. This target vessel is split in two valid target sub-edges by the insertion of a new node, 681 on it. Lastly, the algorithm introduces the erroneous bridge edge 1015, between broken endpoint 680 and introduced target node 681. This results in the final pattern of the perturbation: the original vessel is broken; the

real reconnection is absent of fake reconnection exists. Thus, the appropriate graph repair would be to delete edge 1015 and reconstruct the lost edge between the two endpoints of the break.

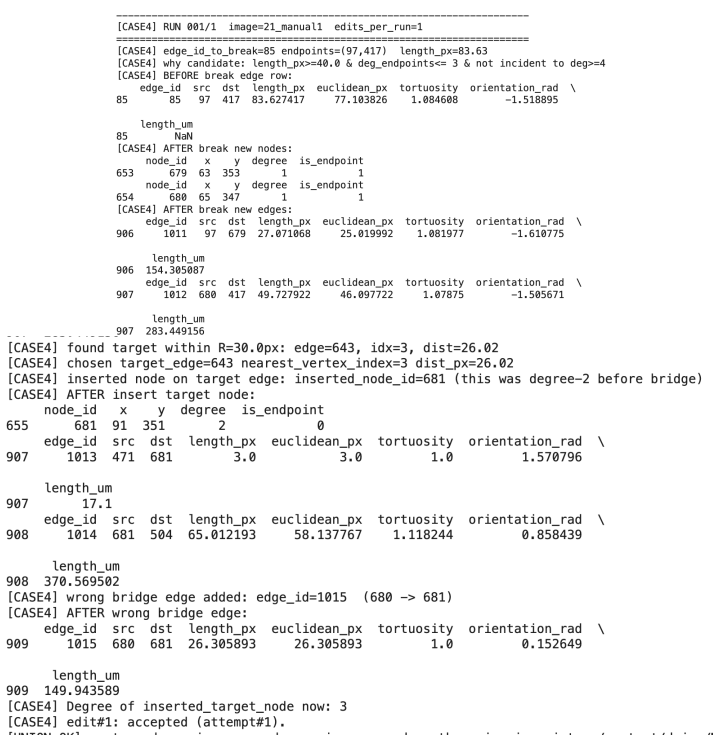


Figure [18]: Example log output of the Case 4 perturbation process illustrating vessel breakage, insertion of intermediate nodes, and incorrect bridging to a nearby vessel.

5.3.4.8. Labels for link prediction in Case 4

Case 4 generates both positive and negative supervision signals.

5.3.4.8.1. Positive missing edge

The two endpoints created by breaking a vessel should be connected in the correct topology. The edge between them is therefore a positive missing edge. It is absent in the perturbed graph but should exist in the corrected structure. The model must learn to restore this missing connection.

5.3.4.8.2. Negative wrong bridge edge

The false bridge connecting a broken endpoint to a nearby vessel is labeled as a negative edge. It exists in the perturbed graph but not in the corrected graph. The model must learn to remove this incorrect connection.

5.3.4.8.3. Removed true edge from the reference

The original vessel edge that was broken is part of the correct topology. Although it is removed during perturbation, it is retained in the union graph as a positive reference edge (in_ref = 1, in_cur = 0, y_edge_exists = 1) to guide the learning process.

5.3.4.9. Meaning of the union graph in Case 4

As in previous cases, supervision is based on a union graph constructed from the corrected reference graph and the perturbed graph.

5.3.4.9.1. Meaning of union nodes

The file nodes_union.csv contains all nodes from both graphs. A node may appear in the reference graph, the perturbed graph, or both.

Node indicators are defined as:

- in_ref = 1: node exists in the reference graph
- in_cur = 1: node exists in the perturbed graph
- is_added = 1: node introduced during perturbation

In Case 4, newly added nodes include: break endpoints and inserted target nodes.

5.3.4.9.2. Meaning of union edges

The file edges_union.csv contains all edges from both graphs, along with labels indicating whether each edge should exist in the corrected topology.

In Case 4, four categories of edges are defined:

1. **Untouched valid edges**

   Present in both graphs

   $\rightarrow y = 1$, label_reason = keep_existing

2. **New valid split edges**

   Created during edge splitting

   $\rightarrow y = 1$, in_ref = 0, in_cur = 1, is_added = 1

3. **Removed true reference edges**

   Present in the reference graph but removed during perturbation

   $\rightarrow y = 1$, label_reason = POS_removed_true_edge

4. **Wrong bridge edges**

   Artificial edges introduced during perturbation

   $\rightarrow y = 0$, label_reason = NEG_wrong_bridge_remove

Virtual Positive Edges

In addition to removed edges, a virtual positive edge is introduced between the two broken endpoints. This edge represents the missing connection that should be restored.

It is defined as:

- in_ref = 1, in_cur = 0, y_edge_exists = 1
- label_reason = POS_missing_uv_virtual

This representation helps the model learn both to remove incorrect edges and restore missing valid connections.

5.3.4.10. Edge Changes in Case 4

5.3.4.10.1. Added edges

- split edges from breaking the original vessel
- split edges from the target vessel
- the wrong bridge edge

Only the bridge edge is invalid.

5.3.4.10.2. Removed edges

- original vessel edge (broken)
- original target edge (split)

The original vessel edge remains part of the correct topology and is preserved in the union graph as a positive reference.

5.3.4.10.2. Missing edge to be repaired

The key missing edge connects the two broken endpoints. Restoring this edge is essential to recover correct vessel continuity.

5.3.4.11.Output files generated for Case 4

After perturbation, the graph is stored as:

- nodes_perturbed.csv

- edges_perturbed.csv
- edgepaths_perturbed.json

Additional metadata and audit files include:

- labels_case4.json
- audit_case4.json
- candidates_edges_case4.csv
- chosen_sites_debug.csv

The union representation is stored as:

- nodes_union.csv, edges_union.csv, edgepaths_union.json

These files support reproducibility and supervised training.

5.4. Mixed Perturbations

As discussed earlier, each perturbation case models a specific type of structural error. However, in real vascular graph extraction, errors rarely occur in isolation. Instead, multiple inconsistencies often appear simultaneously, such as crossing misinterpretations, kissing vessels, dangling spurs, and broken segments.

To better reflect these real-world conditions, this work introduces mixed perturbations, where two or more perturbation types are applied sequentially to the same graph. This approach produces more complex and realistic graph structures, allowing the model to learn how to correct multiple interacting topology errors.

5.4.1 Sequential Perturbation Generation

The corrected reference graph is modified through a sequence of perturbations. Let

$$G_0 = (V_0, E_0)$$

denote the reference graph obtained after graph extraction and correction (Chapter 4).

A sequence of perturbed graphs is generated:

$$G_1, G_2, G_3, G_4$$

where each graph is obtained by applying a perturbation to the previous one, rather than returning to the original reference graph.

For example:

- $G_1 = \text{Case } 1(G_0)$
- $G_2 = \text{Case } 2(G_1)$
- $G_3 = \text{Case } 3(G_2)$
- $G_4 = \text{Case } 4(G_3)$

This sequential design ensures that later perturbations are applied to already modified graphs, resulting in compounded topology errors.

5.4.2 Algorithm for Mixed Perturbation Generation

The mixed perturbation process is applied as follows:

Step 1: Initialize graph

Set $G = G_0$, the corrected reference graph.

Step 2: Apply Case 1 (false crossing)

Crossing perturbations introduce incorrect junctions by connecting vessels that should remain separate.

Update: $G \leftarrow G_1$

Step 3: Apply Case 2 (kissing vessels)

A short artificial bridge is added between nearby vessels.

Update: $G \leftarrow G_2$

Step 4: Apply Case 3 (dangling spur)

A short spurious branch is introduced.

Update: $G \leftarrow G_3$

Step 5: Apply Case 4 perturbation (broken vessel)

A vessel segment is broken, and one endpoint is incorrectly reconnected to a nearby vessel.

Update: $G \leftarrow G_4$

- The final graph contains a mixture of structural inconsistencies, including false crossings, spurious connections, dangling branches, and broken vessels.

5.4.3 Example of Mixed Perturbation Generation

Figure [19]: Example of a mixed perturbation sequence, illustrating how multiple structural errors (crossing misinterpretation, kissing vessels, spur branch, and broken vessel) are progressively introduced into the same graph.

The example starts with the image 21_manual1, in which the initial input is a corrected vascular graph.

Step 1: Case 1 (Crossing perturbation)

In the algorithm, an initial crossing misinterpretation is introduced. The precomputed metadata file crossings meta.json is used to choose a valid crossing location.

There were 26 valid crossing sites in the example in Figure 20 so there was a random selection of one site to place a new junction node at coordinates (318.0, 419.0). The original crossing edges are taken out and four erroneous junction edges set in their place. This

means that the actual crossing connections will lose their positive edges and the new junction edges will be negative edges.

Step 2: Case 2 (Kissing vessels)

The second step is to do a kissing vessel perturbation on the graph generated in Step 1. Two close sections of the vessel are identified with a minimum spacing of around 1 pixel and an angle of approximately 90 degrees, which means that there could be a probable crossing setup. The algorithm adds two new nodes on the two vessels and connects them up using a short bridge edge. In example: 680 and 681 are added as nodes and edge 1015 is added between these two nodes. Such an advantage is a fictitious connection between vessels to be discarded later.

Step 3: Case 3 (Spur generation)

In the third perturbation, a spur artifact is added. One of the candidate edges is chosen and a new node is added to the vessel segment. It is a short branch off this node which forms a hanging spur. In the following diagram: edge 654 is selected, a new split node 682 is added, the former one is substituted by edges 1016 and 1017, a spur node 683 is introduced and a spur edge 1018 is added. The spur node will be a degree-1 node, which reflects a hanging piece of vessels.

Step 4: Case 4 (Broken vessel)

Lastly, the algorithm produces a perturbation of broken vessel. A vessel edge is chosen that is long, and split into two, forming two new break endpoints. In the example: the edge 161 is broken, that is, break endpoints 684 and 685 are generated, the initial edge is exchanged to 1019 and 1020 edges. The algorithm then identifies a neighboring ship and adds a node to it: node 686 gets added on an edge 7, resulting in edges 1021 and 1022. Thereafter a

wrong bridge edge 1023 is formed between node 684 and node 686 and node 685 has not been connected. This adds a false missing node between nodes 684 and 685, in addition to a false bridge edge between node 684 and node 686.

5.4.4 Multiple Variants for Dataset Augmentation

The mixed perturbation process is repeated with alternative random seeds, and parameters to achieve greater training diversity. This yields many corrupted instances of each retinal graph with various combinations of errors that enhance diversity of the dataset and model resilience. This arrangement is detailed in Chapter 7.

5.4.5 Importance of Mixed Perturbations for Learning Topology Repair

Mixed perturbation complicates the process of vessel topology correction since many errors might be present simultaneously. In these circumstances local geometry is frequently not sufficient to establish the appropriate connection. GNNs can solve this by exploiting local and global graph structure to enable the model to discover more trustworthy topology-aware corrections.

CHAPTER SIX
GRAPH NEURAL NETWORK MODEL-BASED LINK PREDICTION FOR TOPOLOGY REPAIR

This chapter presents a framework for predicting vessel connectivity in retinal vessel graphs. The objective is to determine whether a candidate edge between two nodes corresponds to a valid vessel connection in the corrected topology. The method operates on skeletonized vessel graphs and resolves structural ambiguities introduced by perturbations, including broken vessels, false crossings, and spurious junctions.

Two complementary approaches are considered. We first develop a graph neural network (GNN)-based model that learns connectivity patterns from data. In addition, we implement a heuristic geometric baseline to provide a rule-based comparison. Both approaches use the same graph representation and candidate edge set to ensure a fair evaluation.

6.1 Problem Formulation: Edge Existence Prediction

We model topology repair as a binary edge prediction problem. Given a vessel graph $G = (V, E)$, the goal is to determine whether a connection between two nodes should exist in the corrected graph.

For each node pair (u, v), a candidate edge is assigned a label:

$$y_{uv} = \begin{cases} 1 & \text{if a valid vessel connection exists between } u \text{ and } v \\ 0 & \text{otherwise} \end{cases}$$

The model estimates the probability:

$$p_{uv} = P(y_{uv} = 1 \mid G)$$

which represents the likelihood that the candidate edge corresponds to a valid vessel continuation. During inference, a threshold τ is applied to obtain binary predictions.

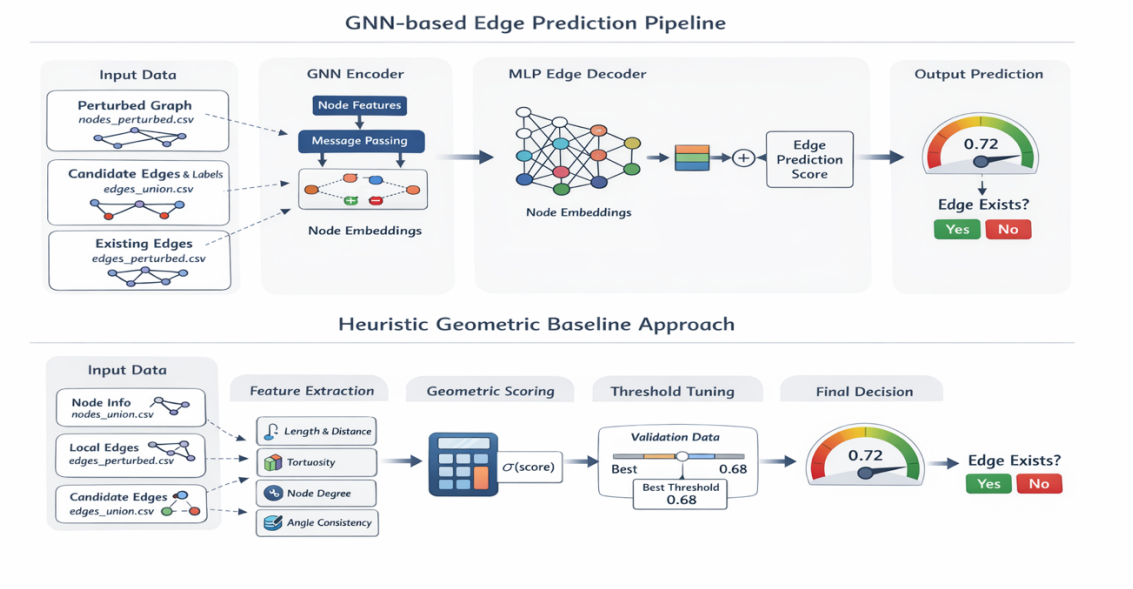


Figure [20]: illustrates the overall pipeline, including both the GNN-based model and the geometric baseline. This formulation enables the model to learn structural patterns that distinguish valid vessel connections from erroneous ones [90].

6.2 Graph Representation and Candidate Edges

The retinal vascular network is represented as a graph where nodes correspond to endpoints and junctions, and edges correspond to vessel segments extracted from the skeletonized image.

The graph is constructed using three main components:

- nodes_perturbed.csv: node coordinates and structural attributes
- edges_perturbed.csv: connectivity of the perturbed graph
- edges_union.csv: candidate edges with supervision labels

The union representation combines existing and candidate edges, allowing the model to learn from both valid and invalid connections. Using the same graph representation for both learning-based and rule-based approaches ensures consistent evaluation.

6.3 Node and Edge Features

We define a set of geometric and structural features to support edge prediction.

Node features include:

- spatial coordinates, node degree, endpoint indicator

Edge features include:

- vessel path length, Euclidean distance, tortuosity, orientation

These features capture both local geometry and structural relationships, enabling the model to distinguish between valid and invalid connections.

6.4 GNN Architecture Framework

We use Graph Neural Networks (GNNs) to learn representations from the vessel graph. GNNs operate on graph-structured data by aggregating information from neighboring nodes through message passing.

The framework follows an encoder–decoder design. The encoder computes node embeddings that capture the structural context of each node. The decoder then predicts the existence of candidate edges based on these embeddings.

We evaluate several architectures, including Graph Convolutional Networks (GCN), GraphSAGE, Graph Attention Networks (GAT), and Variational Graph Autoencoders (VGAE).

GCN aggregates normalized neighbor features to learn node representations.

GraphSAGE introduces learnable aggregation functions and supports inductive learning.

GAT assigns attention weights to neighbors, allowing the model to focus on more informative connections.

VGAE learns probabilistic node embeddings, capturing uncertainty in graph structure.

Each architecture encodes both geometric and structural information into node embeddings.

6.5 Message Passing and Representation Learning

Node embeddings are learned through iterative message passing. At each layer, a node updates its representation by aggregating information from its neighbors.

Let $h_v^{(k)}$ denote the representation of node v at layer k. The update rule is:

$$h_v^{(k)} = \sigma\left(W^{(k)} \cdot \mathrm{AGG}\left(\{h_u^{(k-1)} : u \in \mathcal{N}(v)\}\right)\right)$$

where AGG is an aggregation function, $W^{(k)}$ are learnable parameters, and σ is a nonlinear activation function.

Stacking multiple layers allows nodes to capture information from larger neighborhoods, enabling context-aware representations that are critical for resolving ambiguous connections.

6.6 Edge Prediction Decoder

Given node embeddings, the model predicts whether a candidate edge is valid.

For GCN, GraphSAGE, and GAT, predictions are computed directly from node embeddings by modeling interactions between node pairs.

For VGAE, we use a multilayer perceptron (MLP) decoder. The input is the concatenation of node embeddings:

$$h_{uv} = [z_u \parallel z_v]$$

where z_u and z_v are node embeddings, and ‖ denotes concatenation.

The MLP outputs the probability that the edge exists. This allows the model to capture nonlinear relationships between node features and improves flexibility in modeling complex vessel structures.

6.7 Heuristic Geometric Baseline

We implement a heuristic baseline based on geometric rules. Candidate edges are scored using features such as length, Euclidean distance, tortuosity, and orientation consistency, and classified using a threshold.

While simple and interpretable, this approach relies only on local geometry and does not capture global graph structure. As a result, it is expected to perform хуже than GNN-based models.

CHAPTER SEVEN
EXPERIMENTAL SETUP

7.1 Perturbed Graph Dataset Generation

7.1.1 Source Dataset and Graph Construction

The experiments conducted in this paper have been conducted in DRIVE retinal vessel dataset that comprises of 20 pictures of retinal fundus that are usually used in examining the retinal vessels. The dimensions of the retinal vessels in each image have been symbolized as a graph in a preprocessing pipeline. Vessel segmentation is the initial step in processing the retinal images to extract the vascular structure of the images. The skeletonization of the segmented vessels is then performed to obtain the vessel centerlines. As this representation of the skeleton is a few junction nodes, a few endpoint nodes are computed as well as a graph representation of a straight skeleton is plotted where each node represents a junction point or endpoint point and represents a section of a vessel between two nodes. The outcome of this procedure is an initial graphical model of a retinal vascular net.

7.1.2 Reference Graph Correction

Original graphs derived of the vessel skeleton can have ambiguities in the structure of certain structures especially at the vessel cross point. The process of skeletonization is also not always a guarantee on the existence of a proper path of any node to a skeletonized node and can even cross a vessel. to fix the situation a correction process is carried out to create a graph of the proper graph of a graph of vessels that has a correct topology. This is accomplished by crossing the nodes and the cross point of two vessels which do not share a true common branch is ascertained. Not only does it break up the artificial position of crossings of the type, but it breaks up the different parts of the attacked ships and re-

connects and re-stitches once more at his will. so, to set up the passages of the vessel--no longer a crossing, but two different cross vessels. Fixed graph of any image will be stored with the assistance of the following files: nodes fixed.csv, edges fixed.csv, edgepaths fixed.json and meta corrected crossings meta_corrected.json. Such fixed graphs are perturbed synthetically.

7.1.3 Synthetic Perturbation Strategy

Corrected vessel graphs are perturbed with synthetic perturbations to learn and test models that learn to address structural ambiguities. These noises assume the form of typical topological errors when performing automated vessel extraction. Four cases of perturbation are used:

1. Case 1: Vessel Crossing
2. Case 2: Kissing Vessels
3. Case 3: Dangling Branch (Spur)
4. Case 4: Broken Vessel

In addition to individual perturbation types, mixed perturbation variants are generated by combining multiple perturbations within the same graph to simulate more complex structural errors.

7.1.4 Case 1 — Vessel Crossing

In the first perturbation scenario, false junction nodes are added at vessel crossings, which describes how two vessels can cross each other, but be mistakenly viewed as a branch point. In each corrected graph: 10 trials are carried out; 4 random seeds are varied to

enhance diversity of perturbation. Seeds used: 17, 26, 42, 100. Within each trial, a specific number of crossing sites are chosen: 2, 3, 5, 6, 7, 8, 9, 10, 11, 12. When running a run, previously sampled edges are monitored to make sure that the same point is not sampled more than once in a single trial. This makes generated crossing configurations more varied. On each image: 10 trials multiplied by 4 seeds: 40 samples, Over the 20 images in the DRIVE dataset, $20 \times 40 = 800$ perturbed graphs, across these trials, about 72 crossing edits are used on each image.

7.1.5 Case 2 — Kissing Vessels

The second perturbation example is false vessel connections which is also called kissing vessels. Such connections can be of two kinds, T-shaped junctions, Parallel vessel connections. Multiple experiments are run to produce a variety of examples using varying numbers of edits, and random seeds. The experiments include 10 trials with 20 edits (seed 42), 8 trials with 12 edits (seed 100), 5 trials with 6 edits (seed 10), 2 trials with 2 edits (seed 1000), 2 trials with 25 edits (seed 26). Overall: 27 samples per image, Over the 20 images, there are 27 x 20 = 540 samples of T-shaped kissing vessels. The parallel configuration is repeated on the previously done experiments, and another: 540 samples are produced. Therefore, the size of the set of Case 2 samples is: 1080 Case 2 perturbed graphs.

7.1.6 Case 3 — Dangling Branch (Spur)

The third perturbation scenario adds spurious dangling branches, which are small parts of the vessel that falsely seem to be connected to the vessel network. Spur branches may be injected either: on a given edge, or at a node. Several experiments are carried out using various numbers of edits and random seeds to increase variability.

Table[6]: Experimental configurations for Case 3 (Dangling Branch / Spur), showing the number of trials, injected edits, and random seeds used to generate spur perturbations.

Trials	Edits	Seed
1	3	42
2	60	1
5	10	10
5	20	26
5	25	99
5	40	20
5	50	64
7	15	17
8	30	35
10	12	100

- These experiments result in 53 experiments on each image. There is a total of about 1060 perturbed graphs obtained across the 20 images. They cannot however apply all perturbations since the number of valid candidate locations will vary across images. This means that Case 3 has 848 usable samples in the final dataset. About 265 spur edits are made on each image, when viable locations of injection are available.

7.1.7 Case 4 — Broken Vessel

The fourth perturbation case simulates vessel discontinuities**,** where a continuous vessel segment is artificially disconnected. The following experiments are performed:

Table[7]: Experimental configurations for Case 4 (Broken Vessel), showing the number of trials, injected disconnection edits, and random seeds used to simulate vessel break perturbations.

Trials	Edits	Seed

1	1	100
2	10	10
3	15	26
3	25	99
4	20	17

This results in: 13 trials per image. Across the 20 DRIVE images, the dataset contains: 13 × 20 = 260 perturbed graphs. Each image receives approximately 71 vessel break edits, depending on the availability of suitable vessel segments.

7.1.8 Mixed Perturbation Variants

Several types of perturbations are applied simultaneously to the same graph to produce a more realistic set of graph distortions. It has six different perturbation configurations with each configuration defined by the amount of edits in separate perturbation cases, cross edits, connections that form kissing vessels, spur branches, and broken vessels. All variants are subjected to three random seeds (26, 42, and 100), which gives 18 samples per image (6 variants x 3 seeds). The perturbed dataset is a mixture, with 20 images and gives around 388 perturbed graphs overall. These settings give graphs of different structural complexity allowing the models to be trained on a wide variety of perturbation patterns.

7.1.9 Final Dataset Size

After combining all perturbation cases, the dataset contains:

Table [8]: Number of samples generated for each perturbation case and the mixed perturbation dataset.

Case	**Samples**
Case 1 – Crossing	800

Case 2 – Kissing	1080
Case 3 – Spur	848
Case 4 – Broken	260
Mixed perturbations	388

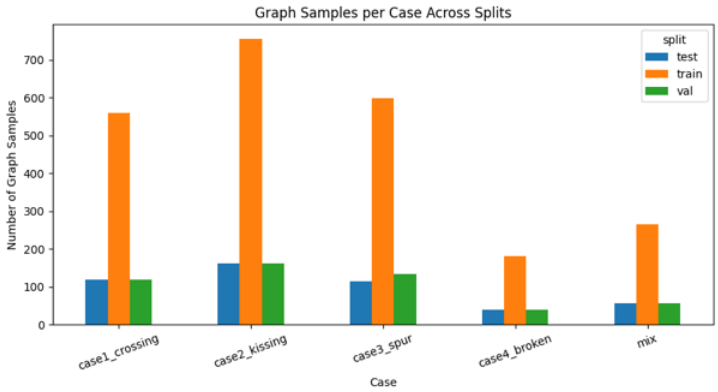
- Total number of perturbed graphs: 3376 samples

7.1.9 Dataset Splitting Strategy

To adequately assess model generalization, the dataset is separated into training, validation and test sets. The splitting is done at the image level as opposed to the graph level. Every retinal image produces numerous distorted samples of graphs. Provided that samples were randomly split on a graph level, perturbations of the same underlying vessel network may occur in both training and testing data. This would enable the model to indirectly view structural patterns in the same image during training, which would result in data leakage and artificially enhanced performance. To avoid this problem, perturbation samples corresponding to a given retinal image are put into the same dataset split. In the DRIVE dataset, there are 20 distinct retinal images, and it is separated into the following sets: Training set: 14 images, Validation set: 3 images, Test set: 3 images. To make it reproducible, a fixed random seed (seed = 42) is used to generate the split.

- Training Images: 22_manual1, 25_manual1, 26_manual1, 27_manual1, 30_manual1, 31_manual1,3 2_manual1, 33_manual1,34_manual1, 35_manual1, 36_manual1, 38_manual1, 39_manual1, 40_manual1
- Validation Images: 23_manual1, 28_manual1, 37_manual1
- Test Images: 21_manual1, 24_manual1, 29_manual1

The below figure 20 illustrates the number of graph samples generated for each perturbation case across the training, validation, and test splits of the dataset.



Figure[21]: Distribution of graph samples across perturbation cases for the training, validation, and test splits.

The graph perturbations in the dataset consist of five categories: crossings of vessels, vessels kissing each other, spur, broken, and mixed perturbations. The distribution indicates that the highest number of samples is in the training set, and the graph neural network models can acquire structural patterns. The sample sizes of both the validation and test set are smaller though proportionally balanced to provide adequate model assessment. Mixed perturbation category indicates more complicated situations in which a series of structural errors are combined, which is a realistic environment to assess the robustness of models.

```
Images per split:
split  num_images
 test           3
train          14
  val           3

Total graph samples per split:
split  num_graphs
 test         494
train        2362
  val         512

Graph samples per case per split:
split           test  train  val
case_name
case1_crossing   120    560  120
case2_kissing    162    756  162
case3_spur       116    598  134
case4_broken      39    182   39
mix               57    266   57
```

Figure [22]: Summary statistics of image and graph sample distribution across dataset splits.

The data used in the experiments are reflected by Figure 21. The data will include 14 training photos, 3 validation photos and 3 test photos which are manipulated to form graph samples. The training, validation and testing samples are 2362 and 512 and 494 respectively. The table also illustrates the number of samples of the graphs provided in each case of perturbation per split. Through the distribution, the entire list of perturbations will be stored in the dataset, and the set of training samples that will be gathered by the model will be enough to make the graph structure.

7.2 Model Training Setup

This section outlines the training setup that is employed to test various measures to make predictions of valid vessel connections of perturbed vessel graphs. A variety of graph neural network (GNN) models along with a geometric heuristic baseline were tested on the same dataset and candidate edge prediction problem. The goal of the models is to indicate the probability that an edge in the union graph is a candidate edge that will give an edge in the corrected topology.

7.2.1 Learning Task

Learning is defined as binary edge existence problem. Each perturbed vessel graph is provided to the model with the perturbed graph structure is going to be used to pass the message, a set of candidate edges, the ground truth labels. All the candidate edges are marked as:

$$y_{edge} = \begin{cases} 1 & \text{valid vessel connection} \\ 0 & \text{invalid candidate edge} \end{cases}$$

The perturbed graph gives the context of the structure, and the supervised learning targets are the candidate edges.

7.2.2 Graph Representation

7.2.2.1. Node Features

The four features calculated as the perturbed graph normalized x coordinate, normalized y coordinate, log-transformed node degree, endpoint indicator are used to represent each node. The nodes are represented with four features derived on the perturbed graph: The node feature vector is thus: $x_i = [x_i, y_i, \log(1 + degree_i), endpoint_i]$. Each graph has its spatial coordinates normalized. The degree of the nodes is log-transformed to normalize its scale. A crucial implementation choice is that the perturbation-specific metadata like is added are not part of node features. Including these attributes would give information on artificially injected nodes and enable the model to use shortcuts during training.

7.2.2.2. Candidate Edge Features

Each candidate edge contains geometric descriptors derived from the vessel graph: vessel length, Euclidean distance between endpoints, Tortuosity, orientation angle. To stabilize learning, the following transformations are applied:

$$[\log(1 + length),\ \log(1 + euclidean),\ tortuosity,\ \sin(\theta),\ \cos(\theta)]$$

These five features form the edge attribute vector used by the edge prediction decoder.

7.2.3 GNN Encoder–Decoder Framework

All neural models follow a common encoder–decoder architecture.

7.2.3.1. Encoder

The encoder computes node embeddings in terms of the perturbed graph: $h = GNN(X, E)$ in which: X is the node features; E is the graph edges in the perturbed graph. The encoder gathers the information of the surrounding nodes in several layers of message passing.

7.2.3.2. Decoder

For each candidate edge (u, v), the decoder constructs an edge representation using node embeddings and edge features:

$$z_{uv} = [h_u \parallel h_v \parallel |h_u - h_v| \parallel h_u \odot h_v \parallel e_{uv}]$$

In this case, h_u and h_v are node embeddings, e u v is the feature vector of the edge, 0 is the concatenation operation, $h_u \odot h_v$ is the element-wise product. These representations are then fed to a multilayer perceptron (MLP) which gives a single logit, the probability of the existence of an edge.

7.2.4 Edge Prediction Pipeline

The prediction process follows a multi-stage pipeline that combines graph representation learning and edge classification. This pipeline is used during both training and inference.

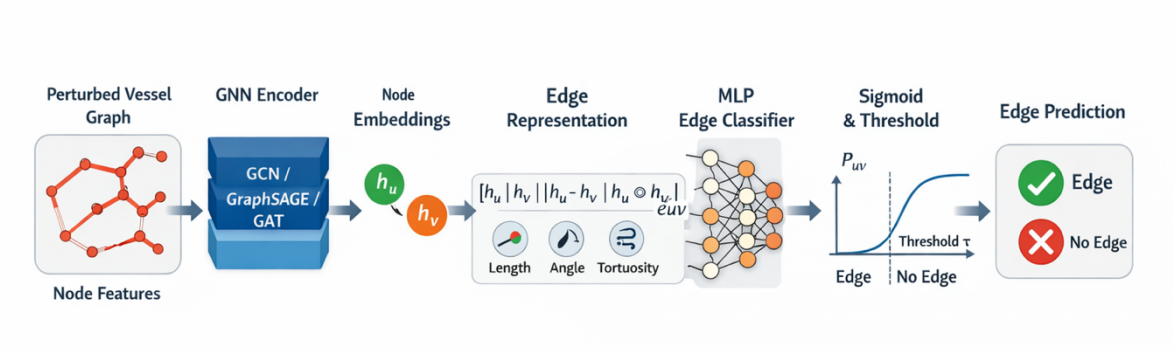


Figure [23]: Edge prediction pipeline used in the proposed graph learning framework. The perturbed vessel graph is first processed by a GNN encoder (GCN, GraphSAGE, or GAT) to generate node embeddings through message passing. For each candidate edge, the embeddings of the two endpoint nodes are combined with geometric edge features to form an edge representation. This representation is passed through an MLP decoder to estimate the probability of edge existence, which is converted to a final prediction using a sigmoid function and a threshold selected on the validation set

The perturbed vessel graph is shown first as a graph (G= V, E) where the nodes are the endpoints or junctions of the vessels, and the edges are the observed vessel connections following perturbation. Each node is linked to feature vector of spatial location and

structural properties. The model starts with the calculation of node embeddings through a graph neural network encoder. Using several levels of message passing, each node is a summation of the information of the adjacent nodes. The node representations at every layer are updated based on the general message passing formulation: The model starts by calculating node embeddings with the help of a graph neural network encoder. The information is aggregated by each node through the various message passing layers of the node with its neighbors. In every layer, a node representation is updated based on the general message passing formulation:

$$h_i^{(k+1)} = \sigma\left(W^{(k)} \cdot \mathrm{AGG}\left(\{h_i^{(k)}, h_j^{(k)} : j \in \mathcal{N}(i)\}\right)\right)$$

In which $h_i^{(k)}$ is the embedding of node i, at layer k, $\mathcal{N}(i)$ is the neighbors of node i, AGG is the aggregation function, is a nonlinear activation function. The learned embedding of every node after the last GNN layer contains both the local and graphical features of the node. Then, an edge representation (a combination of the embeddings of the two endpoint nodes) is built up, with respect to each candidate edge (u, v):

$$z_{uv} = [h_u \parallel h_v \parallel |h_u - h_v| \parallel h_u \odot h_v \parallel e_{uv}]$$

where e_{uv} represents the geometric features of the candidate edge. This edge representation is fed through a multilayer perceptron (MLP) decoder, which generates a single logit value, the probability of the existence of the edge. A sigmoid function is used to convert the logit to a probability: $p_{uv} = \sigma(z_{uv})$.

Finally, a decision threshold is applied to determine the predicted class:

$$\hat{y}_{uv} = \begin{cases} 1 & p_{uv} \geq \tau \\ 0 & p_{uv} < \tau \end{cases}$$

where τ is the classification threshold selected using the validation set. A weighted binary cross entropy loss is used to compare the predicted probabilities with ground truth labels during training. In testing, the trained model will produce probabilities and use the chosen threshold to make final predictions.

7.2.5 Compared Models

Various models were tested to examine the effect of various graph learning methods GCN - Graph Convolutional Network, GraphSAGE - neighborhood aggregation-based GNN, GAT - Graph Attention Network, VGAE + MLP decoder - Variational Graph Autoencoder, GraphSAGE Ablation (no node features), GraphSAGE Ablation (no edge features), Geometric heuristic baseline. These models enable the comparison of learning-based models with the rule based geometric reasoning.

7.2.6 Training Configuration

All neural models were implemented using PyTorch Geometric and trained using GPU acceleration when available. The main training configuration parameters are summarized in Table 9.

Table [9]: Experimental configurations for GNN models

Experiment	Model	Hidden Dim	Layers	Epochs	Batch Size
Exp 1	GCN	64	2	10	4
Exp 2	GraphSAGE	64	2	10	4
Exp 3	GraphSAGE	64	2	10	4
Exp 4	GAT	64	2	15	4
Exp 5	GraphSAGE	128	3	20	4

Exp 6	VGAE + MLP	64	2	10	4
Exp 7	GraphSAGE (no node features)	128	3	20	4
Exp 8	GraphSAGE (no edge features)	128	3	20	4

Table 10: Overall Performance of all evaluated models

Experiment	**Model**	**Batch Size**	**Learning Rate**	**Weight Decay**	**Loss Function**	**Threshold Tuning**
Exp 1	GCN	4	1e-3	1e-4	Weighted BCE	Validation
Exp 2	GraphSAGE	4	1e-3	1e-4	Weighted BCE	Validation
Exp 3	GraphSAGE	4	1e-3	1e-4	Focal Loss	Validation
Exp 4	GAT	4	1e-3	1e-4	Weighted BCE	Validation
Exp 5	GraphSAGE	4	5e-4	1e-4	Weighted BCE	Validation
Exp 6	VGAE + MLP	4	1e-3	1e-4	BCE	Validation
Exp 7	GraphSAGE (no node features)	4	5e-4	1e-4	Weighted BCE	Validation
Exp 8	GraphSAGE (no edge features)	4	5e-4	1e-4	Weighted BCE	Validation

7.2.7 Loss Function and Class Imbalance Handling

The candidate edge dataset exhibits class imbalance, where valid vessel edges significantly outnumber invalid candidate edges. To address this issue, the training loss uses weighted binary cross entropy: $w_{neg} = \frac{N_{positive}}{N_{negative}}$. This increases the penalty for incorrectly predicting the minority class. Some experiments also evaluated focal loss, which emphasizes difficult examples by reducing the contribution of easy predictions [94].

7.2.8 Considerations on Applying SMOTE for Graph-Based Edge Prediction

Although the application of oversampling techniques such as SMOTE (Synthetic Minority Oversampling Technique) is a usual approach in addressing the problem of class imbalance, it was not used in this research. The SMOTE generates artificial examples by interpolating between the ones that are already present in the feature space. Although it works on tabular data, the method presents issues to graph-based vessel structures due to several reasons. First, SMOTE is independent of feature vectors and it does not preserve the topology of the graph. The topological or geometrical invalidity of synthetic samples can occur at the edges of the interpolated sample in the vessel network. Second, the connectivity of the vessels depends on the restriction of spatial relationship and structure which is not justified in generating synthetic edges. Lastly, the use of artificial samples may corrupt the underlying geometric structure that the model has to learn to recognize valid vessel connections. These reasons are why the imbalance problem was solved by loss weighting and threshold tuning, instead of synthetic oversampling.

7.2.9 Threshold Selection

Due to class imbalance, the optimal decision threshold may differ from the default value of 0.5. During training, the classification threshold is tuned on the validation set by evaluating thresholds between 0.05 and 0.95. The threshold that maximizes balanced accuracy is selected. Balanced accuracy is defined as:

$$Balanced\ Accuracy = \frac{Recall_{positive} + Recall_{negative}}{2}$$

This ensures that both valid and invalid edges are predicted reliably.

7.2.10 Geometric Heuristic Baseline

A case-agnostic geometric baseline was applied along with the neural models. This baseline is a geometric plausibility score of each candidate edge which is calculated based on structural and geometric properties, such as endpoint connectivity, node degree, Euclidean distance, vessel length, tortuosity, and alignment with neighboring vessels. A sigmoid function is then used to convert the score into a probability, and a decision threshold is picked on the validation set. This baseline gives a comparison of not learning to determine whether graph neural networks can detect more complicated topological patterns.

7.2.11 Ablation Study

To analyze the contribution of different feature groups, an ablation study was conducted using the GraphSAGE model. Two variants were evaluated:

1. GraphSAGE without node features, where node embeddings are learned without spatial or structural node attributes.

2. GraphSAGE without edge features, where the decoder relies only on node embeddings and ignores geometric edge attributes.

These experiments help determine whether performance improvements arise primarily from graph topology, node attributes, or candidate edge geometry. The detailed results of the ablation study are provided in the appendix.

7.3 Evaluation Metrics

To measure the performance of the proposed models, some classification measures were computed with the predicted labels of candidate edges. Since the problem is introduced as

binary edge existence prediction task, the measurements of the predictions are provided with the assistance of the common binary classification measures with the help of the confusion table. The system of evaluation checks the performance of the model at different levels of granularity so that we are not only aware of the effectiveness of the model in general, but also the degree to which it is resistant to different perturbation scenarios and topology errors.

7.3.1 Confusion Matrix

The predictions are summarized using a confusion matrix to compare them with ground-truth predictions. In edge prediction problem, the confusion matrix is 4-dimensional: True Positive (TP): An edge is in the ground truth, and the model identifies it with being a true edge. False Positive (FP): A candidate edge that is predictable as being valid but is not. True Negative (TN): An edge that has been correctly falsely predicted is called the true negative (TN). False Negative (FN): True and correct connection that is not modeled. In the case of vessel graph reconstruction, especially false negatives are significant because they correspond to vessels connections, which are placed in the fixed vessel topology and are missing.

7.3.2. Evaluation Metrics

Several performance metrics are computed from the confusion matrix [94].

1. Accuracy measures the overall proportion of correctly predicted edges:

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

While accuracy provides a general indication of model performance, it may be misleading when the dataset is imbalanced.

2. Precision measures the proportion of predicted edges that are correct:

$$Precision = \frac{TP}{TP + FP}$$

A high precision indicates that when the model predicts an edge, it is likely to be valid.

3. Recall measures the proportion of true edges successfully identified by the model:

$$Recall = \frac{TP}{TP + FN}$$

Recall is particularly important for this task because it reflects the model's ability to recover missing vessel connections in the perturbed graph.

4. F1 Score is the harmonic mean of precision and recall:

$$F1 = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall}$$

This metric balances the trade-off between precision and recall.

5. Macro F1 averages the F1 score across both classes equally:

$$MacroF1 = \frac{F1_{edge} + F1_{no-edge}}{2}$$

This metric is useful when dealing with imbalanced datasets, since it evaluates performance on both classes independently.

6. Balanced Accuracy measures the average recall across both classes:

$$BalancedAccuracy = \frac{Recall_{edge} + Recall_{no-edge}}{2}$$

Balanced accuracy ensures that the model performs well on both edge predictions and non-edge predictions**,** even when the dataset contains class imbalance.

7.3.3 Multi-Level Evaluation Strategy

To obtain a deeper understanding of model performance, the evaluation is performed at three different levels**.** Each level answers a different question about the behavior of the model.

7.3.3.1. Level 1 Overall Evaluation

The first level evaluates the overall performance of the model across the entire dataset.

All candidate edges from all graphs are aggregated, and the confusion matrix and classification metrics are computed globally. This level answers the question: *"How well does the model predict edge vs. non-edge across the entire dataset?"* This provides a global summary of the model's performance and reflects its overall ability to reconstruct vessel connectivity. Metrics reported at this level include Accuracy, precision, recall, F1 score, macro F1, balanced accuracy.

7.3.3.2. Level 2 Per Sample Case

The second level will compare the performance of each perturbation scenario applied to obtain the dataset, and each graph in the dataset is annotated with a sample_case label of the type of perturbation applied during dataset generation. These cases include case1_crossing, case2_kissing, case3_spur, case4_broken, mix. To conduct this evaluation, the set of graphs in each perturbation scenario has metrics calculated separately. The level will provide the answer to the following question: In case the graph has a certain perturbation situation, how the model can perform on that graph? It measures the stability of the model to various classes of graph perturbations. Notably, this level contains all candidate edges in such graphs, and normal edges and perturbation edges as well.

7.3.3.3. Level 3 Perturbation Edge Analysis

Whereas Level 2 measures the graph-level performance, Level 3 measures the performance of edges which were directly perturbed by the perturbation operations. All the edges are associated with the structural changes that were made when generating the data and they are the most difficult aspect of the issue. Level 3 thus isolates the edges that are the topological errors to the vessel graph. This level provides answer to the question: How detectable are the edges directly related to topology errors by the model? The level 3 is further subdivided into two sub-levels.

7.3.3.3.1. Level 3A Perturbation Edges Only

This evaluation considers only the edges that were created or modified by perturbation operations**.** These edges are identified using the is_perturbation_edge indicator in the dataset. Metrics are computed using only this subset of edges. This level answers the question: *"How well does the model detect edges related to topology errors in the vessel graph?"* Since these edges represent the actual structural mistakes introduced in the graph, this evaluation provides a direct measure of the model's ability to repair graph topology.

7.3.3.3.2. Level 3B Per Edit Type

The last level compares performance of models based on individual perturbation types of edits. Rather than grouping together edges based on the graph situation, the edges are grouped based on the perturbation operation that created it. Some types of edits are crossing edges, kissing vessel edges, spur branches, broken vessel connections and normal edges. The measures are calculated separately on an edit type. The question that this level answers: How effective is the model in distinguishing each kind of topology error? This discussion can be used to discover which structural perturbations are difficult to detect by the model.

7.3.4 Training Monitoring Metrics

Various measures are monitored during epochs of training to detect the learning behavior of the model. This training curves contain training and validation accuracy, training and validation F1 score, training and validation macro F1, training and validation balanced accuracy, training and validation loss.

CHAPTER EIGHT
RESULTS AND ANALYSIS

This chapter shows us the experimental analysis of the suggested graph-based solution to eliminating the topological faults in retinal vessel graphs. The experiments contrast several graph neural network structures with a geometric heuristic baseline to understand their performance in predicting whether edges exist in perturbed vessel graphs.

The major aim of this task is not merely to identify real vessel relationships but also to identify correctly the edges, which are unwanted in the graph. Perturbations can also add spurious edges or remove valid edges in vessel topology correction. Thus, the model should be able to recycle the edges that are not there as well as discard false connections. To assess these capabilities, there were several models under evaluation on a three-level evaluation framework Level 1: Overall model performance, Level 2: Performance in perturbed situations, Level 3A: Edge identification of perturbation, Level 3B: Performance on a single perturbation edit. The models considered are Geometric heuristic baseline, Graph Convolutional Network (GCN), Graph Attention Network (GAT), GraphSAGE (10 epochs), GraphSAGE with focus loss, GraphSAGE (20 epochs), Variational Graph Autoencoder with MLP decoder (VGAE + MLP).

8.1 Overall Model Performance

Model	**Accuracy**	**Recall**	**F1 Score**	**Balanced Accuracy**	**False Negatives**
Heuristic	0.723	0.719	0.836	o.791	154,286
GCN	0.917	0.916	0.956	0.939	45,902
GAT	0.884	0.883	0.937	0.918	64,590
GraphSAGE (10)	0.920	0.919	0.957	0.948	44,757
GraphSAGE + Focal	0.925	0.924	0.960	0.950	41,571

GraphSAGE (20)	**0.942**	**0.942**	**0.970**	**0.953**	**32,043**
VGAE + MLP	0.938	0.937	0.967	0.953	34,379

Table [11] presents the overall performance of all evaluated models.

The results in Table 6 show a clear improvement from the heuristic baseline to graph neural network models for vessel edge prediction. The geometric approach achieves very high precision (0.997) but low recall (0.719), missing many true vessel connections (154,286 false negatives). This highlights the limitation of relying only on local geometric rules. All GNN models significantly improve performance by learning structural patterns. GCN achieves strong results (accuracy 0.917, recall 0.916), greatly reducing missed edges. GAT performs slightly worse (accuracy 0.884, recall 0.883), suggesting limited benefit from attention in this task. GraphSAGE further improves performance. The 10-epoch model achieves accuracy 0.920 and recall 0.919, while adding focal loss increases performance (accuracy 0.925, recall 0.924) by better handling difficult cases. The best results are obtained with GraphSAGE trained for 20 epochs, achieving the highest accuracy (0.942), recall (0.942), and F1-score (0.970), with the lowest number of false negatives (32,043). The VGAE + MLP model also performs strongly (accuracy 0.938, recall 0.937), though slightly below GraphSAGE (20 epochs), and still significantly outperforms the heuristic baseline.

8.2 Confusion Matrix Analysis

To better understand model behavior, Table 7 summarizes the confusion matrix components for all evaluated models.

Model	**TP**	**FP**	**TN**	**FN**
Heuristic	395,557	842	9,390	154,286

GCN	503,941	396	9,836	45,902
GAT	485,253	478	9,754	64,590
GraphSAGE (10)	505,086	236	9,996	44,757
GraphSAGE + Focal	508,272	250	9,982	41,571
GraphSAGE (20)	**517,800**	367	9,865	**32,043**
VGAE + MLP	515,464	325	9,907	34,379

Table [12]: Confusion matrix comparison for all models evaluated on the test dataset.

False Negatives of the heuristic baseline are many and what this means is that geometry rule cannot achieve the complex connectivity of a vessel. The GNN models significantly decrease this problem. In the former, GCN tries to minimize the false negatives to 154, 286 and the GraphSAGE minimized the amounts to 44,757 and 41,571 respectively in the case of focal loss. The highest performance of the GraphSAGE trained and tested on 20 epochs have the minimal false negatives of 32,043. At the same time, the false positives of any GNN model are extremely small i.e. it reconstructs the real edges having the minimal spurious connections. This equilibrium is relevant about the topology correction that is reliable.

8.3 Performance per Perturbation Scenario (Level 2)

The Level 2 evaluation measures performance across different perturbation scenarios present in the dataset.

Model	**Crossing**	**Kissing**	**Spur**	**Broken**	**Mixed**
Heuristic	0.844	0.845	0.503	0.805	**0.508**
GCN	0.925	0.925	0.906	0.910	0.901
GAT	0.891	0.897	0.866	0.885	0.858
GraphSAGE (10)	0.921	0.924	0.914	0.913	0.910
GraphSAGE + Focal	0.928	0.930	0.919	0.922	0.915
GraphSAGE (20)	**0.941**	**0.943**	**0.944**	**0.936**	**0.939**

VGAE + MLP	0.944	0.944	0.929	0.935	0.927

Table [13]: Comparison of model performance across different perturbation cases.

The Level 2 test is used to test the performance of a model on a variety of topology perturbations, such as crossings, kissing vessels, spurs, broken vessels, and mixed errors. Baseline The heuristic is reasonably effective on simpler problems, but is ineffective on harder ones, particularly on spurs and mixed perturbations. Its accuracy in the mixed case is about 0.51 which is near random guessing. On the contrary, the GNN models are robust to every type of perturbation. GraphSAGE trained over 20 epochs is the most stable and consistent in performance amongst them, suggesting that it can learn both geometrical structure and topological structure.

8.4 Perturbation Edge Detection (Level 3A)

Level 3A focuses specifically on edges that are directly affected by topological perturbations.

Model	**Precision**	**Recall**	**F1**
Heuristic	0.713	0.210	0.324
GCN	0.957	0.885	0.920
GAT	0.946	0.833	0.886
GraphSAGE (10)	0.973	0.863	0.915
GraphSAGE + Focal	0.972	0.886	0.927
GraphSAGE (20)	**0.961**	**0.920**	**0.940**
VGAE + MLP	0.966	0.934	**0.951**

Table [14]: Comparison of perturbation edge detection performance (precision, recall, F1-score) across models.

This test is a direct measure of the sensitivity of the model to structural errors added to the graph. The heuristic method is utterly ineffective in this assessment and successfully detects only circa 21 percent of perturbation edges correctly. This proves that the fact that simple geometry rules are unreliably able to detect numerous forms of topology errors. Graph neural networks can largely detect perturbations. GCN model identifies almost 89 percent of perturbation edges whereas GraphSAGE and VGAE models can further enhance this result. The results of GraphSAGE (20 epochs) can detect most perturbation edges with the accuracy of around 92% indicating a strong ability to identify structural irregularities in the vessel graph.

8.5 Performance per Edit Type (Level 3B)

Edit Type	**GraphSAGE (20) Recall**
Crossing	0.898
Kissing	very low
Spur	**0.992**
Broken	0.882
Normal	0.942

Table [15]: Recall of the GraphSAGE (20 epochs) model for perturbation edge detection across different edit types (crossing, kissing, spur, broken, and normal) at Level 3B analysis.

Level 3B provides a detailed analysis of model performance for each perturbation type. Spur perturbations are most readily identified since they create dangling segments which are obviously incompatible with the vessel network. Broken vessels and crossings are more difficult since they entail an inferring of correct connectivity. Kissing vessels are the most difficult as adjacent vessels seem to be joined together when they are not.

8.6 Edge Prediction vs Non-Edge Prediction

In the topology correction task addressed in this thesis, it is important to evaluate not only the ability of the models to detect valid vessel edges, but also their ability to correctly identify edges that should not exist in the graph. Most of the perturbations that have been added to the dataset are related to artificial links in between the segments of the vessels that need to be eliminated to restore a proper vascular topology. To study this behavior, we consider the values of recall of both classes: Edge recall (class 1)**:** ability to recover valid vessel connections and non-edge recall (class 0)**:** ability to correctly reject spurious edges

Model	Edge Recall (Recover True Vessels)	Non-Edge Recall (Reject Spurious Edges)
Heuristic	0.719	0.918
GCN	0.917	0.961
GAT	0.883	0.953
GraphSAGE (10)	0.919	**0.977**
GraphSAGE + Focal	0.924	0.976
GraphSAGE (20)	**0.942**	0.964
VGAE + MLP	0.937	0.968

Table [16] summarizes the recall values for both classes across all evaluated models.

According to the results of Table 11, the difference in the model performance, i.e. differences in the officially rejected model edges and true model edges are statistically significant. The heuristic baseline can perform very well in rejecting edges (non-edge recall 0.918) but also has considerably lower edge recall (0.719) suggesting that the heuristic baseline does not attempt to explore the space of valid association among vessels. The graph neural networks can be significantly recovering edge recovery, but the non-edge detection can be high. The GCN in edge recall is 0.917 and non-edge recall is 0.961 and GAT slightly less edge recall 0.883. Generally, the most optimal of them is the GraphSAGE models. GraphSAGE-10 retrieves the largest amount of non-edge (0.977) and

focal-loss GraphSAGE recalls the largest amount of edges (0.924) and the best performance overall (0.942 and 0.964) in terms of edge and non-edge recall, respectively. VGAE + MLP model is also doing well having the edge recall of 0.937 and non-edge recall of 0.968. Overall, these results support the hypothesis that GNNs can be applied in vessel topology repair because it can rebuild the lost connections, and the fact that the spurious connections can be ignored.

8.7. Discussion of Model Behavior

The findings indicate that graph neural networks do significantly better than geometric heuristics in correcting the vessel topology. The heuristic base assumes principles based on pre-existing geometrical concepts and has issues with more complex structural forms leading to many edges being overlooked. GNNs bypass this by simultaneously learning geometry and structure on graphs through passing of messages [78]. The most successful of the models tried is the GraphSAGE: this is likely because learnable neighborhood aggregation is more effective in approximating the relationship between vessels. Adding more epochs to training GraphSAGE only increases the performance resulting in the highest recall rate and the least amount of edges being missed. In the overall, GraphSAGE is the most promising and accurate solution to overcome the topology errors of retinal vessel graphs.

CHAPTER NINE
FINDINGS AND LIMITATIONS

This chapter summarizes the main experimental findings and discusses the limitations of the proposed approach. The results show that graph neural networks significantly outperform heuristic methods in topology correction; however, the analysis also reveals specific scenarios where performance remains challenging, providing direction for future work.

9.1 Key Findings

9.1.1 Graph Neural Networks Significantly Improve Topology Correction

The results show that graph neural networks outperform rule-based geometric methods in correcting topology errors in retinal vessel graphs. The heuristic baseline relies on fixed geometric rules, which limits its ability to model complex vessel relationships, particularly in ambiguous regions [99].

In contrast, GNNs learn structural patterns directly from the graph, allowing them to incorporate both local geometry and global connectivity. This enables the model to correctly infer vessel continuity even when geometric cues alone are insufficient. The improvement in performance indicates that topology correction is inherently a structural reasoning problem, which cannot be fully addressed by local geometric rules alone.

9.1.2 GraphSAGE Provides the Most Consistent Performance

Among the evaluated models, GraphSAGE achieves the most consistent performance across different metrics and perturbation types. This can be attributed to its neighborhood aggregation mechanism, which effectively balances local and contextual information.

GraphSAGE maintains high recall while limiting false negatives, indicating that it is better at preserving valid vessel connections. This behavior suggests that its inductive learning formulation generalizes well across different graph structures, making it more robust to variations introduced by perturbations.

9.1.3 Graph Neural Networks Effectively Detect Both Edges and Non-Edges

The results demonstrate that GNNs successfully handle both aspects of topology correction: restoring missing edges and removing false connections.

This is a critical property, as topology errors involve both types of mistakes. Models that focus only on detecting existing edges would fail to correct false connections, while models that focus only on removal would not restore missing continuity. The balanced performance across both tasks indicates that GNNs learn a more complete representation of vessel connectivity.

9.1.4 Model Performance Varies Across Perturbation Types

The analysis shows that performance depends on the type of perturbation.

Spur perturbations are easier to detect because they introduce clear structural anomalies, such as short dangling branches.

Broken vessels are more challenging because the model must infer missing continuity without explicit geometric evidence.

Kissing vessels are the most difficult, as nearby vessels exhibit similar geometric features, making it hard to distinguish true and false connections.

This variation highlights that errors requiring global reasoning (e.g., missing connections or ambiguous proximity) are more difficult than those that can be identified through local structure.

9.1.5 Mixed Perturbations Reflect Real-World Conditions

Mixed perturbations provide a more realistic evaluation setting, as multiple topology errors often occur simultaneously in practice. The heuristic baseline performs poorly in this setting because it evaluates each candidate edge independently using local rules. In contrast, GNNs maintain stable performance, suggesting that they can model interactions between multiple structural inconsistencies.

This result highlights an important advantage of learning-based approaches: the ability to reason about interacting errors, rather than treating each error in isolation.

9.2 Limitations

Several limitations exist in the proposed approach of correcting vessel topology errors.

9.2.1 Difficulty in Distinguishing Kissing Vessel Configurations

Kissing vessels remain the most challenging scenario. The difficulty arises because these vessels are spatially close and often have similar orientations, making them indistinguishable based on local geometric features alone.

This limitation suggests that additional contextual information, such as longer-range connectivity or domain-specific anatomical constraints, may be required to improve performance in these cases.

9.2.2 Dependence on Synthetic Perturbations

The training and evaluation rely on synthetic perturbations applied to corrected vessel graphs. While this provides controlled and diverse training data, it may not fully capture the complexity of real segmentation errors.

As a result, the model's performance on real-world data may differ. Future work should evaluate the method on larger datasets with naturally occurring topology errors.

9.2.3 Computational Cost

Training GNNs requires more computational resources than heuristic methods due to iterative message passing and parameter optimization.

However, this cost is primarily incurred during training. Once trained, the model performs inference efficiently. Given the significant improvement in performance, the additional computational cost is justified.

CHAPTER TEN
CONCLUSION AND FUTURE WORK

10.1 Conclusion

This thesis dealt with the issue of repairing topological errors in retinal vessel graphs with graph neural networks. Correct vessel topology is significant in the retinal image analysis process since vessel connection errors may impact the subsequent image processing tasks, including vascular analysis and disease detection. To overcome this issue, the thesis put forward a graph-based model, which predicts the presence of candidate edges indicating a valid vessel connection. The perturbation model was also created to model typical topological errors, such as crossings, kissing vessels, spurs, and broken vessels, with the help of the DRIVE dataset corrected reference graphs. Several models were evaluated, including a geometric heuristic baseline and various graph neural network (GNN) architectures, such as GCN, GAT, GraphSAGE, and VGAE. The results demonstrate that GNNs can learn structural relationships directly from the graph and significantly outperform the heuristic approach. Among these, GraphSAGE achieved the best overall performance, with the 20-epoch variant yielding the highest recall and the fewest false negatives. The findings also revealed the effect of perturbation type on difficulty, where spur errors are the easiest to detect and kissing vessels are the most difficult to detect. In general, the results validate that the graph neural networks are a useful and sound solution to detect and fix the topology errors in retinal vessels graphs.

10.2 Future Work

One way in which the future of the perturbation model could be developed is through the extrapolation to other retina datasets such as HRF, FIVES and CHASE_DB1. This would allow making comparisons between datasets of different image quality, the properties of the vessels, and resolutions that would allow the evaluation of model generalization. In the future, the work can also be expanded to incorporate more classes of perturbations to more effectively model the space of topology errors which are experimentally observed. An alternative approach is to study more advanced models of graph neural networks to fix topology. This experiment was successful with GraphSAGE, although other models like edge-aware GNNs and graph transformers, and specific link prediction algorithms could be more effective to model complex vascular structure, particularly in the difficult cases like kissing vessels.

APPENDIX A – Mixed Perturbation Variants

To model more realistic situations of vessel graph extraction, some more datasets were created with mixed perturbations, i.e. a combination of several types of topology errors in the same graph. These forms are a combination of the perturbation types presented in Chapter 5, such as false crossings (Case 1), kissing vessels (Case 2), dangling spurs (Case 3), and broken vessels (Case 4).

Both variants define the number of perturbation sites used in each type of error. The framework can generate a graph with graphs at different levels of structural complexity by changing the number and mixture of perturbations. This allows testing of the strength of the proposed topology correction model when several topology errors occur at the same time.

Table [17]: Configuration of mixed perturbation variants showing the number of perturbation sites applied for each topology error type.

Variant	Crossing (Case 1)	Kissing (Case 2)	Mode	Spur (Case 3)	Broken (Case 4)
V1	1	1	T	1	1
V2	2	3	Parallel	4	3
V3	4	3	T	2	2
V4	6	4	Parallel	7	8
V5	8	3	T	9	7
V6	4	6	Parallel	10	7

The configurations were applied using randomized candidate selection with fixed seeds to generate multiple perturbed graph instances while maintaining reproducibility.

Appendix B – Code and Dataset Availability

To support reproducibility of the proposed framework, the code used for graph construction, perturbation generation, dataset preparation, and experimental evaluation is made available together with the generated datasets.

Resource	Description
Code to Generate Reference Graphs	Implementation used to construct vessel graphs from skeletonized segmentation masks.
Reference Graphs	Extracted vessel graph representations including nodes, edges, and edge paths.
Code to Generate Corrected Graphs	Implementation of the crossing correction procedure used to refine the initial vessel graphs.
Corrected Graphs	Topologically refined vessel graphs used as the reference structure for perturbation generation.
Code to Generate Case 1 Dataset	Script used to generate false crossing perturbations.
Case 1 Dataset (Crossing)	Synthetic perturbation dataset containing false crossing topology errors.
Code to Generate Case 2 Dataset	Script used to generate kissing vessel perturbations.
Case 2 Dataset (Kissing Vessels)	Dataset containing kissing vessel perturbations with T and parallel configurations.
Code to Generate Case 3 Dataset	Script used to generate dangling spur perturbations.
Case 3 Dataset (Spur)	Dataset containing dangling spur perturbations generated on vessel edges.
Code to Generate Case 4 Dataset	Script used to generate broken vessel perturbations.
Case 4 Dataset (Broken Vessel)	Dataset containing broken vessel segment perturbations.
Code to Generate Mixed Dataset	Script used to generate datasets containing multiple perturbation types simultaneously.
Mixed Perturbation Dataset	Dataset combining multiple topology perturbations within the same vessel graph.
Code to Generate Dataset Splits	Script used to generate the train, validation, and test dataset partitions.
Train / Validation / Test Split Dataset	Predefined dataset split used for training and evaluation experiments.
Experiment Code	Implementation of model training, evaluation, and experimental pipelines.
Heuristic Baseline Model	Implementation of the geometric heuristic baseline used for comparison with GNN models.

Table [18]: Code and dataset resources used in this study for graph construction, perturbation generation, dataset preparation, and model evaluation.

REFERENCES

[1] M. Niemeijer, J. Staal, B. van Ginneken, M. Loog, and M. D. Abràmoff, "Comparative study of retinal vessel segmentation methods on a new publicly available database," in Medical Imaging 2004: Image Processing, vol. 5370, 2004, pp. 648–656. [Online]. Available: https://doi.org/10.1117/12.535349

[2] J. Staal, M. D. Abràmoff, M. Niemeijer, M. A. Viergever, and B. van Ginneken, "Ridge-based vessel segmentation in color images of the retina," IEEE Transactions on Medical Imaging, vol. 23, no. 4, pp. 501–509, 2004. [Online]. Available: https://doi.org/10.1109/TMI.2004.825627

[3] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional networks for biomedical image segmentation," in Medical Image Computing and Computer-Assisted Intervention (MICCAI), 2015, pp. 234–241. [Online]. Available: https://doi.org/10.1007/978-3-319-24574-4_28

[4] P. Liskowski and K. Krawiec, "Segmenting retinal blood vessels with deep neural networks," IEEE Transactions on Medical Imaging, vol. 35, no. 11, pp. 2369–2380, 2016. [Online]. Available: https://doi.org/10.1109/TMI.2016.2546227

[5] R. Estrada, M. J. Allingham, P. S. Mettu, S. W. Cousins, C. Tomasi, and S. Farsiu, "Retinal artery-vein classification via topology estimation," IEEE Transactions on Medical Imaging, vol. 34, no. 12, pp. 2518–2534, 2015. [Online]. Available: https://doi.org/10.1109/TMI.2015.2443117

[6] R. Estrada, S. C. Schmidler, C. Tomasi, and S. Farsiu, "Tree topology estimation," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 37, no. 8, pp. 1688–1701, 2015. [Online]. Available: https://doi.org/10.1109/TPAMI.2014.2382116

[7] A. M. Fraz, P. Remagnino, A. Hoppe, S. A. Velastin, B. Uyyanonvara, A. R. Rudnicka, and S. G. Barman, "Blood vessel segmentation methodologies in retinal images – A survey," Computer Methods and Programs in Biomedicine, vol. 108, no. 1, pp. 407–433, 2012. [Online]. Available: https://doi.org/10.1016/j.cmpb.2012.03.009

[8] M. D. Abràmoff, M. Niemeijer, M. S. A. Suttorp-Schulten, M. A. Viergever, S. R. Russell, and B. van Ginneken, "Evaluation of a system for automatic detection of diabetic retinopathy from color fundus photographs in a large population of patients with diabetes," Diabetes Care, vol. 31, no. 2, pp. 193–198, 2008. [Online]. Available: https://pubmed.ncbi.nlm.nih.gov/18024852/

[9] X. Lyu, L. Cheng, and S. Zhang, "The RETA benchmark for retinal vascular tree analysis," Scientific Data, vol. 9, no. 1, p. 397, 2022. [Online]. Available: https://doi.org/10.1038/s41597-022-01507-y

[10] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in International Conference on Learning Representations (ICLR), 2017. [Online]. Available: https://arxiv.org/abs/1609.02907

[11] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in Advances in Neural Information Processing Systems, 2017. [Online]. Available: https://arxiv.org/abs/1706.02216

[12] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," IEEE Transactions on Neural Networks and Learning Systems, vol. 32, no. 1, pp. 4–24, 2021. [Online]. Available: https://doi.org/10.1109/TNNLS.2020.2978386

[13] N. Patton, T. M. Aslam, T. MacGillivray, I. J. Deary, B. Dhillon, R. H. Eikelboom, K. Yogesan, and I. J. Constable, "Retinal image analysis: Concepts, applications and potential," *Progress in Retinal and Eye Research*, vol. 25, no. 1, pp. 99–127, 2006. [Online]. Available: https://doi.org/10.1016/j.preteyeres.2005.07.001

[14] A. M. Fraz, P. Remagnino, A. Hoppe, S. A. Velastin, B. Uyyanonvara, A. R. Rudnicka, and S. G. Barman, "Blood vessel segmentation methodologies in retinal images—A survey," *Computer Methods and Programs in Biomedicine*, vol. 108, no. 1, pp. 407–433, 2012. [Online]. Available: https://doi.org/10.1016/j.cmpb.2012.03.009

[15] A. F. Frangi, W. J. Niessen, K. L. Vincken, and M. A. Viergever, "Multiscale vessel enhancement filtering," in *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, 1998, pp. 130–137. [Online]. Available: https://doi.org/10.1007/BFb0056195

[16] A. Hoover, V. Kouznetsova, and M. Goldbaum, "Locating blood vessels in retinal images by piecewise threshold probing of a matched filter response," *IEEE Transactions on Medical Imaging*, vol. 19, no. 3, pp. 203–210, 2000. [Online]. Available: https://doi.org/10.1109/42.845178

[17] J. V. B. Soares, J. J. G. Leandro, R. M. Cesar Jr., H. F. Jelinek, and M. J. Cree, "Retinal vessel segmentation using the 2-D Gabor wavelet and supervised classification," *IEEE Transactions on Medical Imaging*, vol. 25, no. 9, pp. 1214–1222, 2006. [Online]. Available: https://doi.org/10.1109/TMI.2006.879967

[18] J. Staal, M. D. Abràmoff, M. Niemeijer, M. A. Viergever, and B. van Ginneken, "Ridge-based vessel segmentation in color images of the retina," *IEEE Transactions on Medical Imaging*, vol. 23, no. 4, pp. 501–509, 2004. [Online]. Available: https://doi.org/10.1109/TMI.2004.825627

[19] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional networks for biomedical image segmentation," in *MICCAI*, 2015, pp. 234–241. [Online]. Available: https://doi.org/10.1007/978-3-319-24574-4_28

[20] P. Liskowski and K. Krawiec, "Segmenting retinal blood vessels with deep neural networks," *IEEE Transactions on Medical Imaging*, vol. 35, no. 11, pp. 2369–2380, 2016. [Online]. Available: https://doi.org/10.1109/TMI.2016.2546227

[21] K. K. Maninis, J. Pont-Tuset, P. Arbeláez, and L. Van Gool, "Deep retinal image understanding," in *MICCAI*, 2016. [Online]. Available: https://doi.org/10.1007/978-3-319-46723-8_17

[22] S. Y. Shin, S. Lee, I. D. Yun, and K. M. Lee, "Deep vessel segmentation by learning graphical connectivity," *Medical Image Analysis*, vol. 58, p. 101556, 2019. [Online]. Available: https://doi.org/10.1016/j.media.2019.101556

[23] Y. J. M. Fateh and M. Rezvani, "VGA-Net: Vessel graph based attentional U-Net for retinal vessel segmentation," *IET Image Processing*, 2024. [Online]. Available: https://doi.org/10.1049/ipr2.13102

[24] K. Liu, S. Gao, Y. Fu, and S. Gao, "Towards generalizable retina vessel segmentation with deformable graph priors," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2025. [Online]. Available: https://openreview.net/forum?id=zVkbsGlKn9

[25] H. Zhao, Y. Sun, and H. Li, "Retinal vascular junction detection and classification via deep neural networks," *Computer Methods and Programs in Biomedicine*, vol. 183, p. 105096, 2020. [Online]. Available: https://doi.org/10.1016/j.cmpb.2019.105096

[26] Á. S. Hervella, J. Rouco, J. Novo, M. G. Penedo, and M. Ortega, "Deep multi-instance heatmap regression for the detection of retinal vessel crossings and bifurcations in eye fundus images," *Computer Methods and Programs in Biomedicine*, vol. 186, p. 105201, 2020. [Online]. Available: https://doi.org/10.1016/j.cmpb.2019.105201

[27] L. K. Pampana and M. S. Rayudu, "Detection and classification of multi-scale retinal junctions using region-based CNN," *Signal, Image and Video Processing*, vol. 16, pp. 265–272, 2022. [Online]. Available: https://doi.org/10.1007/s11760-021-01986-3

[28] J. Long, Y. Xie, N. Que, and P. Yuan, "Retinal vessel junction detection using vessel guidance and heatmap regression," *Measurement*, 2025. [Online]. Available: https://doi.org/10.1016/j.measurement.2025.118966

[29] J. Long, Y. Xie, Y. Zhang, and K. Zhang, "Vessel-guided joint detection and classification of retinal vascular junctions," *Biomedical Signal Processing and Control*, vol. 117, p. 109636, 2026. [Online]. Available: https://doi.org/10.1016/j.bspc.2026.109636

[30] J. Long, Y. Xie, P. Yuan, and Y. Zhang, "Vessel-guided and graph-based retinal vessel junction detection and classification," *Expert Systems with Applications*, vol. 275, p. 126927, 2025. [Online]. Available: https://doi.org/10.1016/j.eswa.2025.126927

[31] G. Wang, Y. Huang, K. Ma, Z. Duan, Z. Luo, P. Xiao, and J. Yuan, "Automatic vessel crossing and bifurcation detection based on multi-attention network vessel segmentation and directed graph search," *Computers in Biology and Medicine*, vol. 155, p. 106647, 2023. [Online]. Available: https://doi.org/10.1016/j.compbiomed.2023.106647

[32] Y. Wang, X. Wang, Z. Gu, W. Liu, W. S. Ng, W. Huang, and J. Cheng, "SuperJunction: Learning-based junction detection for retinal image registration," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, no. 1, pp. 292–300, 2024. [Online]. Available: https://doi.org/10.1609/aaai.v38i1.27782

[33] T. Y. Zhang and C. Y. Suen, "A fast parallel algorithm for thinning digital patterns," *Communications of the ACM*, vol. 27, no. 3, pp. 236–239, 1984. [Online]. Available: https://doi.org/10.1145/357994.358023
[34] O. Jorreia, N. Gonçalves, and R. Cortesão, "Graph-based radiomics feature extraction from 2D retina images," *IEEE Access*, 2025. [Online]. Available: https://doi.org/10.1109/ACCESS.2025.3588817
[35] R. Estrada, M. J. Allingham, P. S. Mettu, S. W. Cousins, C. Tomasi, and S. Farsiu, "Retinal artery-vein classification via topology estimation," *IEEE Transactions on Medical Imaging*, vol. 34, no. 12, pp. 2518–2534, 2015. [Online]. Available: https://doi.org/10.1109/TMI.2015.2443117
[36] Y. Zhao, J. Xie, P. Su, Y. Zheng, Y. Liu, J. Cheng, and J. Liu, "Retinal artery and vein classification via dominant sets clustering-based vascular topology estimation," in *MICCAI*, 2018. [Online]. Available: https://doi.org/10.1007/978-3-030-00934-2_7
[37] F. Girard, C. Kavalec, and F. Cheriet, "Joint segmentation and classification of retinal arteries/veins from fundus images," *arXiv preprint*, 2019. [Online]. Available: https://doi.org/10.48550/arXiv.1903.01330
[38] P. Andreini and S. Bonechi, "Towards a comprehensive characterization of arteries and veins in retinal imaging," *Computers in Biology and Medicine*, vol. 195, p. 110516, 2025. [Online]. Available: https://doi.org/10.1016/j.compbiomed.2025.110516
[39] S. Mishra, Y. X. Wang, C. C. Wei, D. Z. Chen, and X. S. Hu, "VTG-Net: A CNN based vessel topology graph network for retinal artery/vein classification," *Frontiers in Medicine*, vol. 8, p. 750396, 2021. [Online]. Available: https://doi.org/10.3389/fmed.2021.750396
[40] X. Lyu, L. Cheng, and S. Zhang, "The RETA benchmark for retinal vascular tree analysis," *Scientific Data*, vol. 9, p. 397, 2022. [Online]. Available: https://doi.org/10.1038/s41597-022-01507-y
[41] L. Lux, A. H. Berger, M. Romeo Tricas, A. E. Fayed, S. Sivaprasad, L. Kreitner, J. Weidner, M. J. Menten, D. Rueckert, and J. C. Paetzold, "Interpretable retinal disease prediction using biology-informed heterogeneous graph representations," *arXiv preprint*, 2025. [Online]. Available: https://doi.org/10.48550/arXiv.2502.16697
[42] C. Li, L. Lux, A. H. Berger, M. J. Menten, M. R. Sabuncu, and J. C. Paetzold, "Fine-tuning vision language models with graph-based knowledge for explainable medical image analysis," *arXiv preprint*, 2025. [Online]. Available: https://doi.org/10.48550/arXiv.2503.09808
[43] S. Sundar and S. Sumathy, "Classification of diabetic retinopathy disease levels by extracting topological features using graph neural networks," *IEEE Access*, vol. 11, 2023. [Online]. Available: https://doi.org/10.1109/ACCESS.2023.3279393
[44] Q. He, H. Jiang, D. Fang, D. Yang, T. X. Nguyen, A. Ran, C. C. Tham, S. K. H. Szeto, S. Sivaprasad, and C. Y. Cheung, "A graph neural network-based multispectral-view learning model for diabetic macular ischemia detection from color fundus photographs," *arXiv preprint*, 2025. [Online]. Available: https://doi.org/10.48550/arXiv.2502.17886
[45] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, 2009. [Online]. Available: https://doi.org/10.1109/TNN.2008.2005605
[46] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, "Geometric deep learning: Going beyond Euclidean data," *IEEE Signal Processing Magazine*, vol. 34, no. 4, pp. 18–42, 2017. [Online]. Available: https://doi.org/10.1109/MSP.2017.2693418
[47] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *International Conference on Learning Representations (ICLR)*, 2017. [Online]. Available: https://doi.org/10.48550/arXiv.1609.02907
[48] W. L. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *NeurIPS*, 2017. [Online]. Available: https://doi.org/10.48550/arXiv.1706.02216
[49] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *ICLR*, 2018. [Online]. Available: https://doi.org/10.48550/arXiv.1710.10903
[50] T. N. Kipf and M. Welling, "Variational graph auto-encoders," *arXiv preprint*, 2016. [Online]. Available: https://doi.org/10.48550/arXiv.1611.07308
[51] M. Zhang and Y. Chen, "Link prediction based on graph neural networks," in *NeurIPS*, 2018. [Online]. Available: https://doi.org/10.48550/arXiv.1802.09691

[52] S. Shit, J. C. Paetzold, A. Sekuboyina, I. Ezhov, A. Unger, A. Zhylka, J. Pluim, U. Bauer, and B. H. Menze, "clDice—A novel topology-preserving loss function for tubular structure segmentation," in *IEEE/CVF CVPR*, 2021. [Online]. Available: https://doi.org/10.1109/CVPR46437.2021.01629

[53] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph neural networks: A review of methods and applications," *AI Open*, vol. 1, pp. 57–81, 2020. [Online]. Available: https://doi.org/10.1016/j.aiopen.2021.01.001

[54] S. Chen, A. Garcia-Uceda, J. Su, G. van Tulder, L. Wolff, T. van Walsum, and M. de Bruijne, "Label refinement network from synthetic error augmentation for medical image segmentation," *Medical Image Analysis*, vol. 99, p. 103355, 2025. [Online]. Available: https://doi.org/10.1016/j.media.2024.103355

[55] H. Yu, J. Zhao, and L. Zhang, "Vessel segmentation via link prediction of graph neural networks," in *Multiscale Multimodal Medical Imaging (MMMI) Workshop at MICCAI*, 2022. [Online]. Available: https://doi.org/10.1007/978-3-031-18814-5_4

[56] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 1, pp. 4–24, 2021. [Online]. Available: https://doi.org/10.1109/TNNLS.2020.2978386

[57] Y. Ding, Z. Zhang, X. Zhao, D. Hong, W. Cai, C. Yu, N. Yang, and W. Cai, "Multi-feature fusion: Graph neural network and CNN combining for hyperspectral image classification," *Neurocomputing*, vol. 501, pp. 246–257, 2022. [Online]. Available: https://doi.org/10.1016/j.neucom.2022.06.031

[58] P. Yang and X. Zhang, "A dual-branch fusion of a graph convolutional network and a convolutional neural network for hyperspectral image classification," *Sensors*, vol. 24, no. 14, p. 4760, 2024. [Online]. Available: https://doi.org/10.3390/s24144760

[59] J. R. Clough, I. Oksuz, N. Byrne, J. A. Schnabel, and A. P. King, "A topological loss function for deep-learning based image segmentation using persistent homology," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 12, pp. 8766–8778, 2022. [Online]. Available: https://doi.org/10.1109/TPAMI.2020.3013679

[60] Y. Ouyang, H. Zhai, H. Hu, X. Li, and Z. Zeng, "FusionGCN: Multi-focus image fusion using superpixel features generation GCN and pixel-level feature reconstruction CNN," *Expert Systems with Applications*, vol. 262, p. 125665, 2025. [Online]. Available: https://doi.org/10.1016/j.eswa.2024.125665

[61] Y. Zhu, Y. Qiao, Q. Zhou, and X. Yang, "Edge morphology attention mechanism and optimal geometric matching connection model for vascular segmentation," *Biomedical Signal Processing and Control*, vol. 99, p. 106849, 2025. [Online]. Available: https://doi.org/10.1016/j.bspc.2024.106849

[62] S. Go, J. Chae, U. Kim, J. Lim, J. Kim, S. Hogg, E. Trucco, S. J. Park, and S. Lee, "Improving retinal vessel assessment precision by integrating deep learning with interactive editing and graphical modeling," *Scientific Reports*, vol. 15, p. 41552, 2025. [Online]. Available: https://doi.org/10.1038/s41598-025-25421-6

[63] R. Estrada, M. J. Allingham, P. S. Mettu, S. W. Cousins, C. Tomasi, and S. Farsiu, "Retinal artery–vein classification via topology estimation," *IEEE Transactions on Medical Imaging*, vol. 34, no. 12, pp. 2518–2534, 2015. [Online]. Available: https://doi.org/10.1109/TMI.2015.2443117

[64] J. Ortega, J. Novo, M. G. Penedo, and M. Ortega, "Simultaneous segmentation and classification of the retinal arteries and veins from color fundus images," *Artificial Intelligence in Medicine*, vol. 118, p. 102116, 2021. [Online]. Available: https://doi.org/10.1016/j.artmed.2021.102116

[65] S. Chen, A. Garcia-Uceda, J. Su, G. van Tulder, L. Wolff, T. van Walsum, and M. de Bruijne, "Label refinement network from synthetic error augmentation for medical image segmentation," *Medical Image Analysis*, vol. 96, p. 103180, 2024. [Online]. Available: https://doi.org/10.1016/j.media.2024.103355

[66] T. Y. Zhang and C. Y. Suen, "A fast parallel algorithm for thinning digital patterns," *Communications of the ACM*, vol. 27, no. 3, pp. 236–239, 1984. [Online]. Available: https://doi.org/10.1145/357994.358023

[67] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 1, pp. 4–24, 2021. [Online]. Available: https://doi.org/10.1109/TNNLS.2020.2978386

[68] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017. [Online]. Available: https://arxiv.org/abs/1609.02907
[69] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018. [Online]. Available: https://arxiv.org/abs/1710.10903
[70] W. L. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 30, 2017. [Online]. Available: https://arxiv.org/abs/1706.02216
[71] M. D. Abramoff, M. K. Garvin, and M. Sonka, "Retinal imaging and image analysis," *IEEE Reviews in Biomedical Engineering*, vol. 3, pp. 169–208, 2010. [Online]. Available: https://doi.org/10.1109/RBME.2010.2084567
[72] B. Al-Diri, A. Hunter, and D. Steel, "An active contour model for segmenting and measuring retinal vessels," *IEEE Transactions on Medical Imaging*, vol. 28, no. 9, pp. 1488–1497, 2009. [Online]. Available: https://doi.org/10.1109/TMI.2009.2017941
[73] N. Patton et al., "Retinal image analysis: Concepts, applications and potential," *Progress in Retinal and Eye Research*, vol. 25, no. 1, pp. 99–127, 2006. [Online]. Available: https://doi.org/10.1016/j.preteyeres.2005.07.001
[74] T. Y. Zhang and C. Y. Suen, "A fast parallel algorithm for thinning digital patterns," *Communications of the ACM*, vol. 27, no. 3, pp. 236–239, 1984. [Online]. Available: https://doi.org/10.1145/357994.358023
[75] L. Lam, S. W. Lee, and C. Y. Suen, "Thinning methodologies—A comprehensive survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 9, pp. 869–885, 1992. [Online]. Available: https://doi.org/10.1109/34.161346
[76] R. Estrada, M. J. Allingham, and C. Tomasi, "Retinal artery-vein classification via topology estimation," *IEEE Transactions on Medical Imaging*, vol. 34, no. 12, pp. 2518–2534, 2015. [Online]. Available: https://doi.org/10.1109/TMI.2015.2443117
[77] G. Wang, Y. Huang, H. Liu, X. Wang, and Y. Yin, "Automatic vessel crossing and bifurcation detection based on multi-attention network, vessel segmentation and directed graph search," *Computers in Biology and Medicine*, vol. 157, p. 106647, 2023. [Online]. Available: https://doi.org/10.1016/j.compbiomed.2023.106647
[78] W. Chen, S. Yu, K. Ma, W. Ji, C. Bian, C. Chu, L. Shen, and Y. Zheng, "TW-GAN: Topology and width aware GAN for retinal artery/vein classification," *Medical Image Analysis*, vol. 77, p. 102340, 2022. [Online]. Available: https://doi.org/10.1016/j.media.2021.102340
[79] Y. Jalali, M. Fateh, and H. Khotanlou, "VGA-Net: Vessel graph based attentional U-Net for retinal vessel segmentation," *IET Image Processing*, vol. 18, no. 8, pp. 2191–2213, 2024. [Online]. Available: https://doi.org/10.1049/ipr2.13102
[80] C. Shorten and T. Khoshgoftaar, "A survey on image data augmentation for deep learning," *Journal of Big Data*, vol. 6, 2019. [Online]. Available: https://doi.org/10.1186/s40537-019-0197-0
[81] L. Taylor and G. Nitschke, "Improving deep learning using generic data augmentation," *IEEE Symposium Series on Computational Intelligence*, 2018. [Online]. Available: https://doi.org/10.1109/SSCI.2018.8628742
[82] A. Perez and J. Wang, "The effectiveness of data augmentation in image classification using deep learning," *arXiv preprint*, 2017. [Online]. Available: https://doi.org/10.48550/arXiv.1712.04621
[83] Z. Wu et al., "A comprehensive survey on graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 1, pp. 4–24, 2021. [Online]. Available: https://doi.org/10.1109/TNNLS.2020.2978386
[84] W. Hamilton, "Graph representation learning," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 14, no. 3, pp. 1–159, 2020. [Online]. Available: https://doi.org/10.2200/S01072ED1V01Y202003AIM046
[85] T. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *ICLR*, 2017. [Online]. Available: https://doi.org/10.48550/arXiv.1609.02907
[86] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *NeurIPS*, 2017. [Online]. Available: https://doi.org/10.48550/arXiv.1706.02216

[87] P. Veličković et al., "Graph attention networks," *ICLR*, 2018. [Online]. Available: https://doi.org/10.48550/arXiv.1710.10903
[88] T. Kipf and M. Welling, "Variational graph auto-encoders," *NIPS Workshop on Bayesian Deep Learning*, 2016. [Online]. Available: https://doi.org/10.48550/arXiv.1611.07308
[89] M. Zhang and Y. Chen, "Link prediction based on graph neural networks," *NeurIPS*, 2018. [Online]. Available: https://doi.org/10.48550/arXiv.1802.09691
[90] N. A. Holagh and Z. Kobti, "Survey of graph neural network methods for dynamic link prediction," *Procedia Computer Science*, vol. 257, pp. 436–443, 2025. [Online]. Available: https://doi.org/10.1016/j.procs.2025.03.057
[91] J. Niemeijer et al., "Retinopathy online challenge: Automatic detection of microaneurysms," *IEEE Transactions on Medical Imaging*, vol. 29, no. 1, pp. 185–195, 2010. [Online]. Available: https://doi.org/10.1109/TMI.2009.2033909
[92] P. Battaglia et al., "Relational inductive biases, deep learning, and graph networks," *arXiv preprint*, 2018. [Online]. Available: https://doi.org/10.48550/arXiv.1806.01261
[93] D. Powers, "Evaluation: From precision, recall and F-measure to ROC," *Journal of Machine Learning Technologies*, vol. 2, pp. 37–63, 2011. [Online]. Available: https://doi.org/10.48550/arXiv.2010.16061
[94] S. Liu, F. W. Roemer, A. Guermazi, and J. Collins, "Comparison of evaluation metrics of deep learning for imbalanced imaging data in osteoarthritis studies," *Osteoarthritis and Cartilage*, vol. 31, no. 9, pp. 1242–1248, 2023. [Online]. Available: https://doi.org/10.1016/j.joca.2023.05.006
[95] T. Fawcett, "An introduction to ROC analysis," *Pattern Recognition Letters*, vol. 27, no. 8, pp. 861–874, 2006. [Online]. Available: https://doi.org/10.1016/j.patrec.2005.10.010
[96] D. Liben-Nowell and J. Kleinberg, "The link-prediction problem for social networks," *Journal of the American Society for Information Science and Technology*, vol. 58, no. 7, pp. 1019–1031, 2007. [Online]. Available: https://doi.org/10.1002/asi.20591
[97] J. Zhou et al., "Graph neural networks: A review of methods and applications," *AI Open*, vol. 1, pp. 57–81, 2020. [Online]. Available: https://doi.org/10.1016/j.aiopen.2021.01.001
[98] D. Marr and E. Hildreth, "Theory of edge detection," *Proceedings of the Royal Society B*, vol. 207, no. 1167, pp. 187–217, 1980. [Online]. Available: https://doi.org/10.1098/rspb.1980.0020
[99] S. Y. Shin, S. Lee, I. D. Yun, and K. M. Lee, "Topology-aware retinal artery–vein classification via iterative vascular tree estimation," *Applied Sciences*, vol. 11, no. 1, p. 320, 2021. [Online]. Available: https://doi.org/10.3390/app11010320
[100] G. Litjens et al., "A survey on deep learning in medical image analysis," *Medical Image Analysis*, vol. 42, pp. 60–88, 2017. [Online]. Available: https://doi.org/10.1016/j.media.2017.07.005