SAPIENZA
UNIVERSITÀ DI ROMA

Unveiling Protein Connections through Graph Neural Networks and ProtT5 Embeddings

Facoltà di Ingegneria dell'informazione, informatica e statistica

Corso di Laurea in Applied Computer Science and Artificial Intelligence

Candidate

Lorenzo Marinelli
ID number 2043092

Lorenzo Marinelli

Internship Supervisor

Prof. Paola Paci

[illegible]

Academic Year 2024/2025

Unveiling Protein Connections through Graph Neural Networks and ProtT5 Embeddings
Bachelor's thesis. Sapienza – University of Rome

This thesis has been typeset by LaTeX and the Sapthesis class.

Author's email: lorenzo03.marinelli@gmail.com

To the doctors, nurses, and caregivers whose support filled me with determination.

Abstract

Protein–protein interactions (PPIs) underpin nearly all cellular functions, yet many connections in the human interactome remain uncharted because large-scale experiments are slow and expensive. This thesis presents a lightweight graph-neural-network pipeline that combines proteins sequence information encoded as ProtT5 embeddings with network context to predict missing links. Starting from the STRING v12 database—a publicly curated repository of PPIs— only very-high-confidence edges are kept to build a reference graph. The model, trained on commodity hardware, attains an AUROC of 0.96 and retrieves 81% of the withheld positives, while all of the 500 top-ranked predictions correspond to known interactions, confirming excellent early-precision. A focused case study on the NOTCH2 protein further shows that the network suggests biologically plausible partners that warrant wet-lab validation. Overall, the results demonstrate that compact, interpretable GNNs can accelerate interactome mapping and help direct experimental efforts toward the most promising protein pairs.

Contents

Chapter 1

Introduction, Background & Related Work

1.1 Biological importance of protein–protein interactions

Proteins rarely act in isolation. Inside every living cell they assemble into dynamic complexes whose coordinated behavior drives all molecular processes, from DNA replication to signal transduction and energy metabolism. As soon as two or more proteins interact, they create emergent capabilities that do not exist in the individual components: catalytic centers form, mechanical forces are generated, and information is routed through. Modern high-throughput screens and large-scale computational surveys suggest that a typical eukaryotic cell contains millions of distinct protein–protein interactions (PPIs) [1], Although which PPIs occur can vary with cellular conditions (different tissues or environments may activate different interaction subsets),so mapping these context-specific interactions remains an open challenge in proteomics[2]

Health, disease and therapeutics. Because PPIs sit at strategic control points, their disruption can have far-reaching consequences. Single-point mutations that alter an interaction interface may propagate through entire pathways, contributing to cancer, neuro-degeneration or developmental disorders. Conversely, deliberately modulating specific PPIs has emerged as a promising therapeutic route, expanding the notion of what is considered curable with drugs. As condition-specific interactomes are still beyond experimental reach and research techniques remain noisy and costly, computational prediction plays a critical role in closing the gap. Accurate models enable:

- **Hypothesis generation** by identifying plausible interaction partners to guide experimental design, saving time and resources.
- **Network analysis** with algorithms: centrality, clustering, community detection—operate on predicted edges to reveal functional modules and disease mechanisms.
- **Drug discovery** by virtual screening against predicted interfaces expanding the chemical search space and highlighting novel intervention points.

- **Synthetic biology** to design new pathways or molecular machines which require reliable forecasts of how engineered proteins will interact within the host cell.

In summary, protein–protein interactions constitute the organizational scaffold of the cell, translating genetic information into biological function. For computer scientists, they represent a rich, structured prediction problem: nodes are proteins, edges are interactions, and the goal is to infer a large, dynamic, complex network from limited, noisy observations.

1.2 PPI–prediction techniques and limitations

A recent community benchmark of twenty-six algorithms[3] offers a concise snapshot of the current state of the art in network-based protein–protein-interaction (PPI) prediction. Five broad methodological families emerge:

(1) Similarity-based methods. These algorithms assign a likelihood of interaction from pre-defined topological scores— for example, Common Neighbors, Jaccard, Resource Allocation, Katz distance, or the "L3" motif count. They are parameter-free and extremely fast, making them attractive baselines and useful components in larger ensembles.

(2) Probabilistic methods. Stochastic block models and their variants assume that real interactomes contain latent communities; intra-community edges are more probable than inter-community edges. Model parameters are learned by maximizing the likelihood of the observed graph, yielding posterior interaction probabilities that are often interpretable in terms of functional modules.

(3) Factorisation-based methods. Matrix-, tensor- or spectral-factorization techniques compress the high-dimensional adjacency matrix into low-rank node embeddings that preserve global wiring patterns. Unseen PPIs are scored either by cosine similarity in embedding space or by a shallow classifier trained on the compressed representations.

(4) Machine-learning methods. Graph-neural-network (GNN) frameworks dominate this category. Message-passing layers aggregate sequence, edge and neighbourhood features to learn rich node embeddings, which a decoder then converts into edge probabilities. Hybrid architectures incorporating both sequence language-model embeddings and graph structure are steadily improving accuracy.

(5) Diffusion-based methods. Random-walk-with-restart, heat-kernel and other diffusion processes propagate information through the interactome, capturing higher-order context.

Together these families span a spectrum from lightweight heuristics to deep, data-hungry models. Their complementary strengths explain why competitive benchmarks consistently select representatives from each group when building practical PPI-prediction pipelines.

1.3 Graph neural networks in bioinformatics

Graphs provide a natural language for biological data: small molecules are atom–bond graphs, cellular networks link genes, proteins, metabolites and phenotypes, and patient cohorts can be represented as graphs. Graph neural networks (GNNs) offer a principled way to learn from such structures by propagating and transforming feature information along edges and have already achieved promising results in various bioinformatics problems [4].

Message-passing paradigm. Most GNN architectures iterate over the same sequence of steps: each node gathers *messages* from its neighbours, aggregates them with a permutation-invariant operator (summation, mean, attention, max...), and updates its hidden state through a learnable function. Stacking L layers yields a receptive field that covers the L-hop neighborhood, allowing the model to capture local chemistry or long-range dependencies.

Key bioinformatics tasks:

Link prediction: Inferring missing edges between proteins, drugs and targets, or gene–disease associations.

Node classification: Assigning functions to unannotated genes or predicting residue roles in binding interfaces.

Graph classification: Estimating molecular properties,comorbidities of patients, toxicity or compound efficacy from whole-molecule graphs.

Generative modelling: Designing novel proteins or small molecules by learning latent graph distributions.

Practical considerations. Biological graphs are typically **sparse but noisy**, exhibit **power-law degree** and may contain multiple edge types or directions [5]. The remainder of this thesis adopts a message-passing encoder tailored to the topology of protein–protein interaction graphs, coupling pre-trained sequence embeddings with task-specific pairwise decoders to recover missing edges at proteome scale.

1.4 Public PPI data sets (STRING, BioGRID, IntAct)

Reliable computational models of protein–protein interactions require large, uniformly formatted repositories of experimentally supported edges. The three databases summarised below constitute primary examples of that; each exhibits a distinct balance between breadth, annotation depth and curation strategy.

STRING. The STRING database integrates heterogeneous evidence—laboratory assays, comparative genomics, co-expression/co-occurrence analyses and automated text mining—into a single confidence score ranging from 0 (no support) to 1000 (maximal support).[6] Release 12 comprises more than 20 billion scored associations covering approximately 15 000 organisms. The availability of a continuous confidence value facilitates threshold selection, class-rebalancing and weighted-loss designs in supervised learning pipelines.

BioGRID. BioGRID is populated exclusively through manual curation of primary literature.[7] For each interaction, curators record the experimental technique, throughput category and PubMed identifier, yielding high-precision but comparatively sparse graphs.

IntAct. The IntAct repository adopts the PSI-MI (Proteomics Standards Initiative Molecular Interaction) schema and therefore provides detailed, standardised metadata for every entry: host organism, tagging strategy, affinity matrix, detection instrument and additional protocol variables.[8] IntAct forms part of the IMEx consortium, whose members exchange curation records to minimise redundancy and maximise coverage.

Rationale for data-set selection. STRING serves as the source of positive edges in this work owing to its extensive coverage and quantitative scoring, as well as the direct alignment between STRING identifiers and the ProtT5 sequence embeddings used as node features.

1.5 Protein language models and ProtT5 embeddings

Large transformer models can be trained on unlabeled amino-acid sequences with the same masked-token objective that underpins BERT and T5 in natural-language processing. The encoder learns context-dependent representations in which elements that share structural or functional roles occupy similar regions of latent space. By averaging those embeddings, one obtains a fixed-length vector per protein that can serve as an informative node feature in graph neural networks.

SPACE: pre-computed ProtT5 embeddings. Running ProtT5[9] on a full proteome demands substantial GPU memory and computation time. The SPACE resource distributes pre-computed ProtT5 embeddings for all proteins present in the STRING interaction network, stored under the same stable STRING identifiers.[10] This work loads those vectors directly, eliminating the inference step and guaranteeing one-to-one alignment between node features and graph vertices.

In summary, ProtT5 supplies protein embeddings (derived by protein sequences) that integrate seamlessly with the STRING-based graph used in this study.

1.6 Graph representation learning: GCN, GraphConv, GAT, TAG, TransformerConv, GIN

Graph neural networks (GNNs) learn node or graph embeddings by iteratively exchanging information along the edges of a graph. The layers reviewed below span the main families now available in `PyTorch Geometric` (PyG) described to give an overview of the different architectures,some of which will later constitute the encoder design explored in chapter 3. For clarity, let $\mathbf{X} \in R^{|\mathcal{V}| \times F}$ denote the matrix of node features, $\mathbf{A}$ the adjacency matrix, and $\mathcal{N}(i)$ the one-hop neighbourhood of node i.

1. Graph Convolutional Network (GCN)

Kipf and Welling started from the spectral definition of graph convolution, where a signal $\mathbf{x}$ is filtered in the graph–Fourier domain using the eigenbasis of the Laplacian $\mathbf{L} = \mathbf{D} - \mathbf{A}$. Because directly computing eigenvectors is infeasible for large graphs, they proposed a first-order Chebyshev approximation that collapses to a single sparse matrix multiplication [11]:

$$\mathbf{X}' = \hat{\mathbf{D}}^{-1/2}\,\hat{\mathbf{A}}\,\hat{\mathbf{D}}^{-1/2}\,\mathbf{X}\,\mathbf{\Theta}, \qquad \hat{\mathbf{A}} = \mathbf{A} + \mathbf{I}, \tag{2.1}$$

where $\mathbf{X} \in R^{|V| \times d_{\text{in}}}$ is the node-feature matrix and $\mathbf{\Theta} \in R^{d_{\text{in}} \times d_{\text{out}}}$ is the learnable weight matrix.

Why add self-loops? The identity matrix $\mathbf{I}$ ensures that each node also contributes its own features to the average, acting as a residual connection that stabilises deep stacks of GCN layers. Without self-loops information would be lost after the first hop for isolated vertices.

Node-wise view. Writing out the i-th row gives the intuitive message-passing form used in PyG's implementation [12]:

$$\mathbf{x}_i' = \mathbf{\Theta}^\top \sum_{j \in \mathcal{N}(i) \cup \{i\}} \frac{e_{j,i}\,\mathbf{x}_j}{\sqrt{\hat{d}_j\,\hat{d}_i}}, \tag{2.2}$$

where $\hat{d}_i = 1 + \sum_{j \in \mathcal{N}(i)} e_{j,i}$ is the degree after self-loop insertion and $e_{j,i}$ is an optional edge weight (default 1). The expression makes it clear that GCN performs a *normalised mean* over the immediate neighbourhood followed by a shared linear transform.

Limitations. Because information is blended symmetrically, the vanilla GCN cannot differentiate a lot between distinct nodes that share the same 1-hop degree profile, and its fixed, radius-one receptive field requires stacking multiple layers to capture long-range dependencies—issues later addressed by models such as TAGConv and GATConv.

2. GraphConv

`GraphConv` (sometimes referred to as the "improved Weisfeiler–Lehman convolution' [13]) relaxes several constraints of the vanilla GCN while keeping the computational footprint linear in the number of edges.

Separate self– and neighbour transforms. Instead of mixing all messages through a single weight matrix, the layer applies two independent affine maps

$$\mathbf{x}_i' = \underbrace{\mathbf{W}_{\text{self}}\mathbf{x}_i}_{\text{root signal}} + \underbrace{\mathbf{W}_{\text{nbr}} \sum_{j \in \mathcal{N}(i)} e_{j,i}\,\mathbf{x}_j}_{\text{aggregated neighbours}}, \tag{2.3}$$

which allows the network to treat information that already resides at the node differently from information imported from its surroundings. If $\mathbf{W}_{\text{self}} = \mathbf{W}_{\text{nbr}}$ and $e_{j,i} = 1/\sqrt{\hat{d}_i \hat{d}_j}$ one recovers the GCN update as a special case.

Expressive advantage. Because $\mathbf{W}_{\text{self}} \neq \mathbf{W}_{\text{nbr}}$ the layer can already distinguish two nodes that share identical one-hop neighbourhoods but differ in their own features—a limitation of the original GCN.

Limitations and positioning. GraphConv is still a radius-1 operator: long-range information requires stacking or skip connections. It also remains permutation equivariant, meaning it cannot encode absolute node positions as TransformerConv does. In this thesis we use GraphConv only for baselines; TAGConv and attention layers provide larger receptive fields with similar runtime.

3. Graph Attention Network (GAT)

Veličković *et al.* replaced the fixed, degree-normalised averaging of GCN with a data–driven weighting scheme that lets the network decide which neighbours deserve attention [15]. Starting from node features $\mathbf{X} \in R^{|V| \times F}$, the layer first projects every vector through a shared linear map $\mathbf{\Theta} \in R^{F \times F'}$. The projected descriptors are then paired and scored by a small single-layer network—nothing more than a dot product with two learnable vectors $\mathbf{a}_s, \mathbf{a}_t \in R^{F'}$ followed by a LeakyReLU with fixed slope 0.2. Because the scores are passed through a soft-max over the 1-hop neighbours of the source node, the coefficients

$$\alpha_{i,j} = \text{softmax}_j\Big(\text{LeakyReLU}(\mathbf{a}_s^\top \mathbf{\Theta} \mathbf{x}_i + \mathbf{a}_t^\top \mathbf{\Theta} \mathbf{x}_j)\Big) \tag{2.4}$$

form a proper probability distribution $\sum_{j \in \mathcal{N}(i) \cup \{i\}} \alpha_{i,j} = 1$. A node update is therefore the expectation of its transformed neighbours

$$\mathbf{x}'_i = \sum_{j \in \mathcal{N}(i) \cup \{i\}} \alpha_{i,j} \, \mathbf{\Theta} \, \mathbf{x}_j, \tag{2.5}$$

which recovers plain mean aggregation if the network happens to assign uniform weights[16].

Why is this useful? GCN implicitly treats every neighbour as equally informative once degree normalisation is applied. In protein–interaction graphs that assumption is shaky: some partners represent well-studied core complexes, others are single low-evidence mentions in a text-mining channel. By letting $\alpha_{i,j}$ depend on the feature content of *both* nodes, GAT can suppress noisy edges and amplify biologically relevant ones without any handcrafted filtering.

Limitations. Scalar attention is a powerful upgrade over the uniform averaging of GCN, but it still comes with two practical caveats. First, stacking many GAT layers drives all node embeddings toward a common sub-space, because every message is a weighted *copy* of the same linearly projected features. Second, the mechanism cannot refine a message *after* the weight is applied: once the scalar α_{ij} is set, the neighbour contributes $\alpha_{ij} \, x_j$ and nothing more.

4. Topology–Adaptive Graph Convolution (TAGConv)

Topology–Adaptive Graph Convolution (TAGConv) was introduced by Du *et al.* as a vertex-domain alternative to the Chebyshev polynomial filters used in spectral GNNs [17].The operator expresses a graph convolution as a polynomial of the symmetrically-normalised adjacency, thereby capturing multi-hop context without stacking many layers.

Definition. Let $\mathbf{A}$ be the adjacency matrix (self-loops optional) and $\mathbf{D}$ the diagonal degree matrix. Define the normalised adjacency $\hat{\mathbf{A}} = \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$. A single TAGConv layer [18] computes:

$$\mathbf{X}' = \sum_{k=0}^{K} \hat{\mathbf{A}}^{k} \mathbf{X} \mathbf{W}_k, \tag{2.6}$$

where $\hat{\mathbf{A}}^0 = \mathbf{I}$ copies the node itself and higher powers mix progressively larger k-hop shells. The weight matrices $\{\mathbf{W}_k\}$ act as a **filter bank**: during training the network learns how much emphasis each hop should receive. Equation (2.6) implements a K^{th}-order polynomial filter whose coefficients adapt during training, yielding local receptive fields analogous to dilated kernels in CNNs [17].

Intuition. Equation (2.6) is the graph analogue of a dilated K-tap FIR filter in 1-D signal processing. Instead of stacking K separate GCN layers, TAGConv aggregates $0-K$-hop information in parallel and fuses it in a single matrix product. TAGConv supplies a computationally efficient, multi-scale aggregation rule whose polynomial filter bank offers both theoretical clarity and practical performance advantages compared to multiple GCNConv or GraphConv layers, for large, sparse graphs such as protein–protein interaction networks.

5. TransformerConv

TransformerConv [19] adapts the scaled dot–product attention mechanism of the Transformer encoder [20] to graph data in `PyTorch Geometric`[21]. Given input features $\mathbf{X} \in R^{|\mathcal{V}| \times F}$, the layer proceeds as follows.

Core update rule. For each node i the output is a skip connection plus an attention-weighted sum of neighbour messages:

$$\mathbf{x}'_i = \mathbf{W}_1 \mathbf{x}_i + \sum_{j \in \mathcal{N}(i)} \alpha_{i,j} \mathbf{W}_2 \mathbf{x}_j, \tag{2.7}$$

where $\mathbf{W}_1, \mathbf{W}_2 \in R^{F \times F'}$ are learnable projections.

Attention coefficients. Multi-head dot-product attention computes the mixing weights

$$\alpha_{i,j} = \text{softmax}_j \left(\frac{(\mathbf{W}_3 \mathbf{x}_i)^\top (\mathbf{W}_4 \mathbf{x}_j)}{\sqrt{d}} \right), \tag{2.8}$$

with $\mathbf{W}_3, \mathbf{W}_4 \in R^{F \times d}$. The layer parameter `heads` (H) controls how many independent replicas of (2.8) run in parallel; their outputs are concatenated when `concat=True` and averaged otherwise.

TransformerConv therefore combines the expressive power of transformer attention with linear complexity in the number of edges, and its optional edge-feature and gating mechanisms make it particularly suitable for heterogeneous, confidence-weighted PPI graphs.

6. Graph Isomorphism Network (GINConv)

The GINConv[22, 23] operator matches the discriminative power of the 1-Weisfeiler–Lehman test by employing an injective multi-layer perceptron after neighbourhood aggregation.

Update rule. Let $\mathbf{X} \in R^{|\mathcal{V}| \times F}$ be the node-feature matrix and $\mathbf{A}$ the (unweighted) adjacency with self-loops removed. A single GINConv layer computes

$$\mathbf{X}' = h_{\mathbf{\Theta}}\Big((\mathbf{A} + (1+\epsilon)\mathbf{I})\mathbf{X}\Big), \tag{2.9}$$

where $h_{\mathbf{\Theta}} : R^F \to R^{F'}$ is an arbitrary MLP defined by the user, and $\epsilon \in R$ controls the weight assigned to each node's own features relative to the summed neighbour signal.

Relation to other layers. Equation (2.9) reduces to a one-hop `GraphConv` when $h_{\mathbf{\Theta}}$ is linear and $\epsilon = 0$, but the universal function approximation property of MLPs endows GIN with strictly higher expressive power than fixed aggregators such as GCN.

1.7 Link–prediction evaluation metrics (AUROC, AUPRC, $P@K$, NDCG)

Predicting missing edges in a graph yields a **ranked list** of candidate links. Performance is therefore assessed both as a **threshold–independent** summary of the score distribution (AUROC, AUPRC) and as a **top-K** ranking quality (precision at K, NDCG). The metrics reviewed below are implemented in `scikit-learn` and `PyTorch Geometric`; their mathematical definitions follow the respective library documentation.

Notation. Let $\mathcal{P} = \{(u,v)_+\}$ and $\mathcal{N} = \{(u,v)_-\}$ denote the sets of positive and negative edges in a test split, and let $s_{uv} \in R$ be the model score assigned to pair (u,v). A descending sort of all candidate pairs $\mathcal{P} \cup \mathcal{N}$ yields a ranked list $\langle (u_1,v_1), (u_2,v_2), \dots \rangle$.

1. Area under the ROC curve (AUROC)

The receiver–operating–characteristic curve plots the true–positive rate $\mathrm{TPR}(\tau)$ against the false–positive rate $\mathrm{FPR}(\tau)$ as the decision threshold τ sweeps over R. AUROC is the Riemann integral $\int_0^1 \mathrm{TPR}(f)\, df$, equal to the probability that a random positive edge is scored higher than a random negative one.Values range from 0.5 (chance) to 1 (perfect). AUROC is threshold- independent but can be overly optimistic under extreme class imbalance, as often encountered in dense link-prediction scenarios.

2. Area under the Precision–Recall curve (AUPRC)

Precision $\mathrm{P}(\tau) = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FP}}$ and recall $\mathrm{R}(\tau) = \frac{\mathrm{TP}}{|\mathcal{P}|}$ are computed for each τ; connecting the points yields the PR curve whose integral is AUPRC. Because the baseline is directly influenced by the positive class prevalence, AUPRC is more sensitive than AUROC to correctly measuring the performance of retrieving the rare positive edges in highly imbalanced graphs.

3. Precision at K (P@K)

For the top-K items of the ranked list,

$$P@K \;=\; \frac{1}{K}\sum_{i=1}^{K} \mathbf{1}[(u_i, v_i) \in \mathcal{P}]. \tag{2.10}$$

P@K answers the practical question "How many of the first K suggestions are correct?", mirroring retrieval and recommender-system usage.Typical choices in PPI studies are $K = 100$, 500, matching wet-lab validation budgets.

4. Normalised Discounted Cumulative Gain (NDCG)

Discounted cumulative gain at K is DCG@$K = \sum_{i=1}^{K} \frac{g_i}{\log_2(i+1)}$, with $g_i \in \{0, 1\}$ the relevance of item i. Dividing by the ideal DCG yields NDCG@$K \in [0, 1]$.The log discount weights early ranks more heavily than later ones, rewarding algorithms that place true interactions near the top of the list.

5. Accuracy

Accuracy measures the fraction of correctly classified edges when a single decision threshold is applied to the score distribution:

$$\mathrm{ACC}(\tau) = \frac{\mathrm{TP}(\tau) + \mathrm{TN}(\tau)}{|\mathcal{P}| + |\mathcal{N}|}. \tag{2.11}$$

In practice, τ is set to 0.5 after a sigmoid or softmax activation. Although intuitive, accuracy can be misleading under severe class imbalance: when positives represent only 1 % or less of all candidate pairs, a trivial all-negative classifier already achieves 99 % accuracy. For this reason, accuracy is reported in **supplementary tables** only and not used for model selection; threshold-independent (AUROC, AUPRC) and top-K ranking metrics (P@K, NDCG) provide a more sensitive assessment of link-prediction quality.

6. Metric selection in this thesis

AUROC and AUPRC quantify overall separability, independent of K.

P@500 reports the wet-lab hit-rate for a realistic batch size.

NDCG@500 penalises near-misses more smoothly than P@K, providing a gradient-friendly signal during ablation studies.

Together these four metrics capture complementary aspects of link-prediction quality—ranking fidelity over the full score range (AUROC, AUPRC) and actionable precision within the experimentally accessible top slice (P@K, NDCG).

Chapter 2

Data Pre-processing

2.1 STRING v12: download, filtration, confidence thresholds and mapping to ProtT5 embeddings

This thesis relies on the high-confidence **human** subset of STRING v12.0.[1] The data-preparation pipeline converts the original text and HDF5 files into a `PyTorch Geometric` graph whose nodes carry fixed ProtT5 feature vectors. Every step is fully deterministic.

1. Acquisition

- **Interaction list** `9606.protein.links.v12.0.txt.gz`: three columns (`protein1`, `protein2`, `combined_score`).
- **ProtT5 embeddings** `9606.protein.sequence.embeddings.v12.0.h5`: datasets `/proteins` (N gene ID strings) and `/embeddings` ($N \times 1024$ `float32`).

2. Confidence filtering

Only interactions with `combined_score` ≥ 950 are retained. This threshold preserves the top of STRING evidence and yields a graph size that fits comfortably into GPU memory while minimising false positives from lower-score edges.

3. Graph construction

1. **Node indexing** All unique STRING identifiers in the filtered edge list are collected and mapped to consecutive integers via `pandas.factorize`. The resulting node count is
$$|V| = 10\,430.$$
2. **Edge indexing** Each protein pair (u, v) is translated to an undirected edge (i, j) between the corresponding integer indices. Duplicate pairs and self-loops are discarded, leaving
$$|E| = 117\,118$$
The final edge list is stored in the `edge_index` tensor of shape $(2, |E|)$ required by `torch_geometric.data.Data` for the creation of the dataset used by the GNN.

[1] Links and embeddings were retrieved from the official FTP mirror in May 2025

4. Feature matrix

1. Strip the taxon prefix (`9606.`) from each STRING ID, build a hash map `id` $\mapsto$ `row` into the HDF5 file, and copy the associated 1 024-dimensional ProtT5 vector.
2. Assemble the feature tensor $\mathbf{X} \in R^{|V| \times 1024}$; missing entries (none in this release) would be zero-filled.

5. Resulting data object

The resulting `Data` instance contains

- `x` $= \mathbf{X}$ (node features),
- `edge_index` $= 117\ 118$ undirected edges,
- `num_nodes` $= 10\ 430$.

This immutable graph $G = (V, E)$ forms the input to all subsequent train/validation/test splits and model experiments discussed.

2.2 Train/validation/test splits with `RandomLinkSplit` and negative sampling

Link prediction is evaluated on edges **not present** during training. Accordingly, the high-confidence STRING graph assembled in Section 2.1 is split into three disjoint subsets with `torch_geometric.transforms.RandomLinkSplit`.[24]

1. Protocol

1. **Positive-edge partition**: All 117 118 undirected interactions are shuffled once with a fixed seed and divided in the ratio 80:10:10 % $\rightarrow$ training, validation, test.
2. **Graph masking** Validation and test positives are **removed** from the adjacency used for message passing. Thus, during both training and inference, the GNN must infer these links from the remaining network context and from what it has learnt from the training set.

2. Nomenclature: message-passing graph vs. label sets

Before describing splits it helps to pin down the two objects that appear in every link-prediction experiment:

Message-passing graph $G^{\mathbf{msg}} = (V, E^{\mathbf{msg}})$. : The adjacency on which all GNN layers operate. It is constructed **once per epoch** by taking the training positives and sampling training negatives. No positive validation or test edge ever enters E^{msg}.

Label sets : Tuples (`edge_label_index`, `edge_label`) that the network scores. They contain *both* positives and dynamically sampled negatives; only these pairs contribute to the loss or evaluation metrics.

3. Resulting data splits

	MP edges	Pos. label edges	Neg. label edges
Training ($\mathcal{E}^{\text{train}}$)	93 698	46 849	on–the–fly at 1:10 ratio
Validation ($\mathcal{Q}^{\text{val}}$)	93 698	5 855	58 550 (fixed, 1:10)
Test ($\mathcal{Q}^{\text{test}}$)	105 408	5 855	58 550 (fixed, 1:10)

Table 2.1. Edge counts for training, validation, and test splits

Each *query* split—validation and test—is stored as

$$\texttt{edge\_label\_index} \in N^{2\times|\mathcal{Q}|}, \qquad \texttt{edge\_label} \in \{0,1\}^{|\mathcal{Q}|}.$$

4. Why this design?

- **Better Generalization.** Validation and test edges are fully hidden during message passing. The GNN must infer them from network context, not memorisation.
- **Realistic class imbalance.** A 1 : 10 positive–negative ratio approximates more closely the sparsity of the interactome and makes evaluation statistics more meaningful.

2.3 Negative-sampling strategies and class-imbalance considerations

Why sample negatives? The number of absent edges in a protein–protein interactome is orders of magnitude larger than the known positives. Storing every non-edge is infeasible and would bias the loss excessively toward the majority class. Sampling provides a tractable, tunable approximation.

Uniform random sampling (baseline). At the beginning of each epoch for every positive $(u, v) \in \mathcal{E}^{\text{train}}$, ten node pairs $(u', v') \notin E$ are drawn uniformly at random. This yields $\approx 9.4 \times 10^5$ negative examples per epoch.

Loss weighting. A common remedy for $1:10$ class imbalance is to weight the binary cross-entropy terms by the inverse class frequency, i.e. $w_+ = 1$ for positives and $w_- = 0.1$ for negatives. In this study the weights are *not* applied. Leaving the loss unbalanced biases the decision boundary toward delivering lower scores on average, which translates into higher **precision** in the top-ranked predictions—the regime that matters most when only a limited number of candidates can be validated experimentally. The accompanying drop in recall is acceptable given the model priority on reliability rather than exhaustive coverage.

Chapter 3

Methods

3.1 Overview of the encoder–decoder architecture

Let $G = (V, E)$ be the high-confidence STRING graph defined in Chapter 2. Each vertex $v \in V$ carries a fixed ProtT5 feature vector $\mathbf{x}_v \in R^{1024}$; each undirected edge $(u, v) \in E$ denotes a confirmed interaction. Link prediction is cast as a supervised scoring problem: given two vertices $u, v \in V$ the model returns a logit $\ell_{uv} \in R$ whose sigmoid $\sigma(\ell_{uv}) \in [0, 1]$ estimates the probability that u and v interact.

The architecture follows the *encode–process–decode* pattern (Fig. 3.1):

1. **Encoder** $\mathcal{E} : R^{|V|\times 1024} \times N^{2\times|E^{\text{msg}}|} \to R^{|V|\times d}$ propagates information along the *message-passing graph* $E^{\text{msg}} \subseteq E^{\text{train}}$ and outputs a d-dimensional embedding $\mathbf{z}_v = \mathcal{E}(\mathbf{X}, E^{\text{msg}})_v$ for every vertex.

2. **Pair decoder** $\mathcal{D} : R^d \times R^d \to R$ consumes two vertex embeddings and returns the interaction logit $\ell_{uv} = \mathcal{D}(\mathbf{z}_u, \mathbf{z}_v)$.

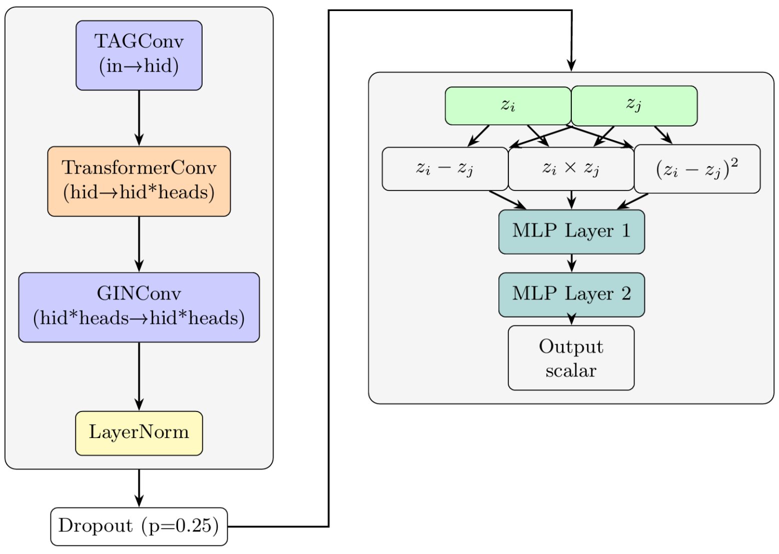


Figure 3.1. This modular design offers two advantages: Training and inference on $\mathcal{D}$ are **edge-local**; and task-specific strategies can be explored with the decoder and encoder independent of each other.

3.2 Encoder stack

The encoder maps the fixed 1024-dimensional ProtT5 vectors $\mathbf{x}_v^{(0)}$ to a task-specific embedding $\mathbf{z}_v \in R^d$ by applying three successive message-passing layers, each followed by ReLU and—for the final output—layer normalisation (Fig. 3.1). Hyper-parameters were tuned under the constraint that the full model, graph and mini-batch negatives fit in less than 16 gb of Vram.

Layer	**Input dim**	**Output dim**
TAGConv ($K = 3$)	1024	192
TransformerConv (6 heads)	192	1152
GINConv	1152	1152
LayerNorm	1152	1152

Table 3.1. Dimensions of the Encoder Stack layers.

3.2.1 TAGConv ($K = 3$) for multi-hop context

The first layer broadens each node's receptive field by aggregating messages from its 1-, 2- and 3-hop neighbourhood in a *single* matrix operation [17]:

$$\mathbf{X}^{(1)} = \sum_{k=0}^{3} (\hat{\mathbf{A}})^k \mathbf{X}^{(0)} \mathbf{W}_k, \quad \hat{\mathbf{A}} = \mathbf{D}^{-1/2} \mathbf{A} \mathbf{D}^{-1/2}, \tag{3.1}$$

where $\mathbf{W}_k \in R^{1024\times192}$ form a learnable polynomial filter bank. Choosing $K = 3$ balances coverage and memory.

Why $K = 3$ TAGConv? The topology–adaptive convolution offers a hyper-parameter K that controls how many hops of the normalised adjacency $\hat{\mathbf{A}}$ are summed. Setting $K = 3$ is motivated by both biological and graph-theoretic considerations.

Biological "three-hop" rule. Network biology studies consistently report that function, pathway membership and phenotype similarity decay sharply beyond three interaction steps. Capturing paths longer than three therefore adds diminishing biological signal while increasing the risk of aggregating unrelated neighbourhoods.

Avoiding over-smoothing. Stacking multiple one-hop GCN layers is a naïve way to enlarge the receptive field, but empirical work shows that more than three–four layers cause node representations to converge to a subspace where they become indistinguishable ending up with **over-smoothing**. TAGConv with $K = 3$ attains a three-hop receptive field in a *single* layer, limiting depth to three total graph convolutions (TAG $\rightarrow$ Transformer $\rightarrow$ GIN) and thus reducing over-smoothing risk.

Choosing $K = 3$ strikes a balance between biological reach, representation sharpness and hardware efficiency: it captures nearly all functionally relevant paths, avoids the over-smoothing that plagues deeper stacks of one-hop layers, and keeps the encoder small enough for consumer-grade GPUs.

3.2.2 Multi-head TransformerConv

The second layer refines the coarse TAGConv representation using graph transformer attention [21]. For six heads ($H = 6$) and hidden width $d_h = 192$:

$$\alpha_{i,j}^{(h)} = \text{softmax}_j\Big(\frac{(\mathbf{W}_Q^{(h)}\mathbf{x}_i^{(1)})^\top(\mathbf{W}_K^{(h)}\mathbf{x}_j^{(1)})}{\sqrt{d_h}}\Big) \tag{3.2}$$

$$\mathbf{x}_i^{(2)} =_{h=1}^{H} \sum_{j\in\mathcal{N}(i)} \alpha_{i,j}^{(h)}\,\mathbf{W}_V^{(h)}\mathbf{x}_j^{(1)}, \tag{3.3}$$

followed by a residual projection $\mathbf{W}_{\text{root}}\mathbf{x}_i^{(1)}$ Concatenating heads yields $6 \times 192 = 1152$ features per node. This choice of heads and hidden dimention was taken empirically during different trials with varying number of heads and dimensions with same Gpu Vram usage.

Why attention? Preliminary trials with `GraphConv`, `PNAConv`, `ResGatedGraphConv` and `GATConv` showed TransformerConv as the better option; probably because of its ability to modulate neighbor importance by leveragine the query/key/value logic of the Transformer, which is particularly valuable for hub-dominated and noisy PPI graphs, where naïve mean aggregation dilutes signal or propagates irrelevant information.

3.2.3 GIN fine-tuning layer and normalisation

A final `GINConv` layer increases expressivity by applying an injective MLP after neighbour summation [22]:

$$\mathbf{X}^{(3)} = \text{MLP}_\Theta\Big[(1+\epsilon)\mathbf{I} + \mathbf{A}\Big]\mathbf{X}^{(2)},, \tag{3.4}$$

Why a final GINConv layer? The last encoder stage is a single `GINConv` layer applied after TAG- and Transformer-based aggregation. GINConv employs an *injective* multi-layer perceptron after neighbour summation. Placed at the end of the stack it serves the purpose of adding more expressivity while keeping computation costs low, as it acts as a final non-linear transformation to polish the representation.

3.3 Pair decoder

Once vertex embeddings $\mathbf{z}_u, \mathbf{z}_v \in R^d$ have been produced by the encoder, the **Pair Decoder** maps the ordered pair (u, v) to a scalar logit $\ell_{uv} \in R$ that is transformed into a link probability via $\sigma(\ell_{uv})$. The decoder is *edge-local*: it uses no graph structure and can score any pair on demand, a property exploited during negative mining and large-scale inference.

4.3.1 Feature construction

Given two nodes features We construct a *feature vector*

$$\mathbf{h}_{uv} = \big[\underbrace{\mathbf{z}_u - \mathbf{z}_v}_{\text{difference}} \;\|\; \underbrace{\mathbf{z}_u \odot \mathbf{z}_v}_{\text{Hadamard}} \;\|\; \underbrace{(\mathbf{z}_u - \mathbf{z}_v)^2}_{\text{squared diff.}}\big] \in R^{3d}. \tag{3.5}$$

Why three operators? *Difference* encodes relative orientation in latent space, *Hadamard* captures element-wise similarity, and the *squared difference* emphasises large coordinate gaps.

4.3.2 Multi-layer perceptron

The final predictor is a two-layer MLP (without biases)

$$\ell_{uv} = f_{\mathbf{\Theta}}(\mathbf{h}_{uv}) = \mathbf{W}_2\,\sigma(\,\mathrm{BN}(\mathbf{W}_1\mathbf{h}_{uv})), \tag{3.6}$$

Norm and ReLU are used after each linear layer. the first MLP takes in input nodes with R^{3d} features, (reminder: $d = 192 * 6\,heads = 1152$) and outputs $d/6 = 192$, the second MLP then projects these features to a scalar.

3.4 Parameter count of the model

Table 3.2 enumerates the trainable parameters of the architecture in Fig. 3.1.

Component	**Formula**	**# params**
TAGConv (K+1 = 4)	$(K{+}1)\,1024 \times 192{+}192$	786624
TransformerConv (H=6 heads)	$(H \times 192 \times 192 + 1\,152) \times 4$	889344
GINConv	1152×1152	1327105
LayerNorm (affine)	2×1152	2304
Encoder subtotal		3005377
Pair MLP (3456 → 192 → 1)	$3456 \times 192 + 192 \times 1$	663744
Total model		3669121

Table 3.2. Trainable parameter breakdown

3.5 Loss function and class weighting

Link prediction is formulated as a binary classification problem on candidate edges. For every queried pair (u, v) the decoder outputs a logit $\ell_{uv} \in R$; applying the logistic map $p_{uv} = \sigma(\ell_{uv})$ produces a probability estimate $p_{uv} \in (0, 1)$ that the two proteins interact. The training signal is obtained from the *binary cross-entropy* criterion

$$\mathcal{L} = -\frac{1}{|\mathcal{B}|} \sum_{(u,v)\in\mathcal{B}} \Big[y_{uv} \log p_{uv} + (1 - y_{uv}) \log(1 - p_{uv})\Big], \tag{3.7}$$

where $\mathcal{B}$ is the mini-batch of positive and sampled negative pairs and $y_{uv} \in \{0, 1\}$ is the ground-truth label.

Class imbalance. Each epoch draws 10 negatives for every positive ($|\mathcal{B}|_{\text{neg}} : |\mathcal{B}|_{\text{pos}} = 10{:}1$). As stated previously: although inverse-frequency weighting would equalise the two terms in Eq. (3.7), it was *not* employed. Leaving the loss unweighted biases the optimisation towards conservative scores.

Implementation. Equation (3.7) is realised with `torch.nn.BCEWithLogitsLoss`, which internally applies the sigmoid transform and is numerically more stable for large $|\ell_{uv}|$ compared to a sigmoid followed by a normal binary cross entropy loss.

With this setup the model learns to assign high probabilities only when the combined sequence and topology evidence is compelling, thereby maximising the reliability of the small set of pairs that will be taken forward for experimental validation.

3.6 Edge-dropout graph augmentation

Graph-neural networks tend to overfit when the message-passing graph is sparse but *deterministic*: the model can memorise short paths between frequently co-occurring vertices, inflating validation metrics while failing to generalise to edges that connect previously unlinked regions. To counteract this effect the training graph is *stochastically thinned* each epoch by removing a fixed proportion of its edges.

DropEdge operation. Let E^{train} be the positive edge set available for message passing (Section 2.2). Before the first forward pass of every epoch, a Bernoulli mask $\mathbf{m} \in \{0,1\}^{|E^{\text{train}}|}$ is sampled i.i.d. with

$$\Pr[m_e = 0] = p_{\text{drop}}, \qquad \Pr[m_e = 1] = 1 - p_{\text{drop}},$$

and only the retained edges $E^{\text{msg}} = \{e \in E^{\text{train}} \mid m_e = 1\}$ are fed to the encoder. A value of $p_{\text{drop}} = 0.60$ was chosen empirically. DropEdge affects only message passing; the *labels* used in the loss (Equation (3.7)) remain unchanged, so the model must infer certain positive edges without directly "seeing" them in the graph context of that epoch.

Expected effects. Edge dropout serves two roles:

Regularisation. Randomly removing certain edges forces neighbouring nodes to rely on alternative multi-hop paths, reducing the dependence of the model on specific paths

Data augmentation. Over successive epochs the GNN is exposed to an ensemble of graph realisations $G^{(1)}, G^{(2)}, \ldots$, analogous to dropout in CNNs; the resulting vertex embeddings approximate an expectation over local topological perturbations.

3.7 Optimiser and learning-rate schedule

Optimiser choice. All experiments use the `AdamW` optimiser. AdamW combines adaptive first- and second-moment estimates with *decoupled* weight decay.

Hyper-parameter	Symbol	Value
Initial learning rate	η_0	5×10^{-6}
Weight decay	λ	1×10^{-5}
β-coefficients	(β_1, β_2)	(0.9, 0.999)

Table 3.3. Optimizer hyper-parameters

Cosine–annealing schedule (no restarts). The learning rate is updated with PyTorch's `CosineAnnealingLR`.

Here $\eta_0 = 5 \times 10^{-6}$ is the initial learning rate and $\eta_{\min} = 0$ is the floor. With `T_max` = 3000 equal to the planned number of epochs, the *per-epoch* update rule is

$$\eta_t = \eta_{\min} + \tfrac{1}{2}(\eta_0 - \eta_{\min})\left[1 + \cos(\pi\, t/3000)\right], \quad t = 0, 1, \ldots, 2999. \tag{3,8}$$

Thus the step size declines smoothly over the entire course of training with no abrupt drops or restarts.

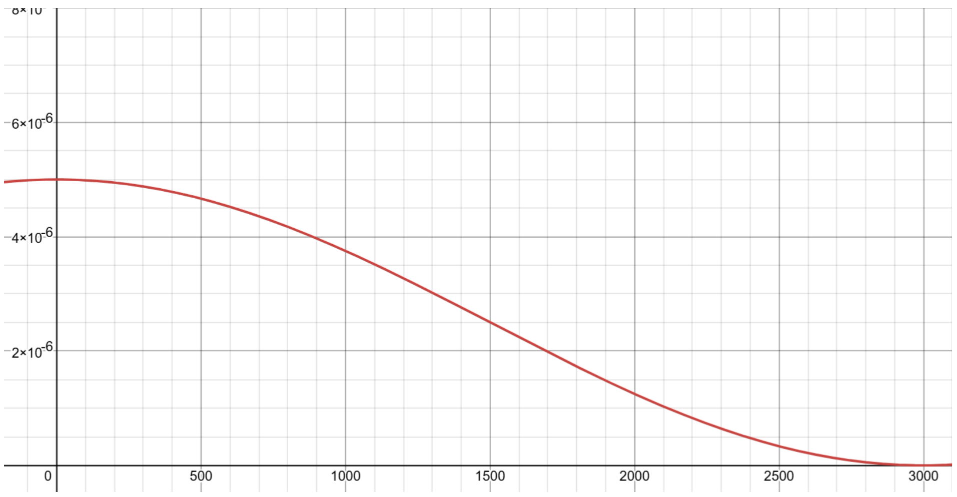


Figure 3.2. Learning rate over 3000 epochs

3.8 Compound z-score checkpointing (AUPRC + P@500)

Selecting the "best" model by a single metric can be misleading: area–under–PR (AUPRC) captures global separability, whereas precision-at-500 (P@500) reflects the reliability of the small candidate set destined for wet-lab validation. To balance both objectives the training loop monitors a *compound z-score* that standardises each metric against the performance of a random ranker and then sums the two normalised values.

4.8.1 Random-ranker baseline

For a validation set of n edge pairs with positive prevalence p_{pos}:

$$\begin{aligned} \mu_{\text{AUPRC}} &= p_{\text{pos}}, \quad \sigma_{\text{AUPRC}} = \sqrt{\frac{p_{\text{pos}}(1-p_{\text{pos}})}{n}}, \\ \mu_{P@K} &= p_{\text{pos}}, \qquad \sigma_{P@K} = \sqrt{\frac{p_{\text{pos}}(1-p_{\text{pos}})}{K}}, \end{aligned} \tag{3.9}$$

where $K = 500$. Equation (3.9) follows from the fact that both AUPRC and $P@K$ reduce to the sample mean of n (or K) Bernoulli trials when scores are uniformly random.

4.8.2 Per-epoch normalisation

At the end of each epoch the model produces validation scores $\widehat{\mathrm{AUPRC}}_t$ and $\widehat{P@500}_t$. These are converted to standard scores

$$z_{\mathrm{AUPRC},t} = \frac{\widehat{\mathrm{AUPRC}}_t - \mu_{\mathrm{AUPRC}}}{\sigma_{\mathrm{AUPRC}}}, \qquad z_{P@500,t} = \frac{\widehat{P@500}_t - \mu_{P@500}}{\sigma_{P@500}}, \tag{3.10}$$

and aggregated into a single statistic

$$\boxed{Z_t = z_{\mathrm{AUPRC},t} + z_{P@500,t}}. \tag{3.11}$$

Their sum rewards epochs that simultaneously push both global separability and top-K precision beyond random expectation.

4.8.3 Checkpoint criterion

The model state with the highest Z_t seen so far is checkpointed:

$$\texttt{if } Z_t > Z_{\mathrm{best}}\texttt{:} \quad \texttt{save(model)}$$

Early stopping is *not* triggered automatically; training proceeds for the full 3 000 epochs, but only the best-Z weights are retained for test-set evaluation.

Empirical behaviour. Across different random seeds the compound score typically peaks between epochs 2500 and 3000 epochs, together with the other statistics. Using Z_t therefore selects models with slightly lower recall but markedly higher P@500—a trade-off consistent with the project's focus on delivering a small, high-confidence interaction listfor experimental validation.

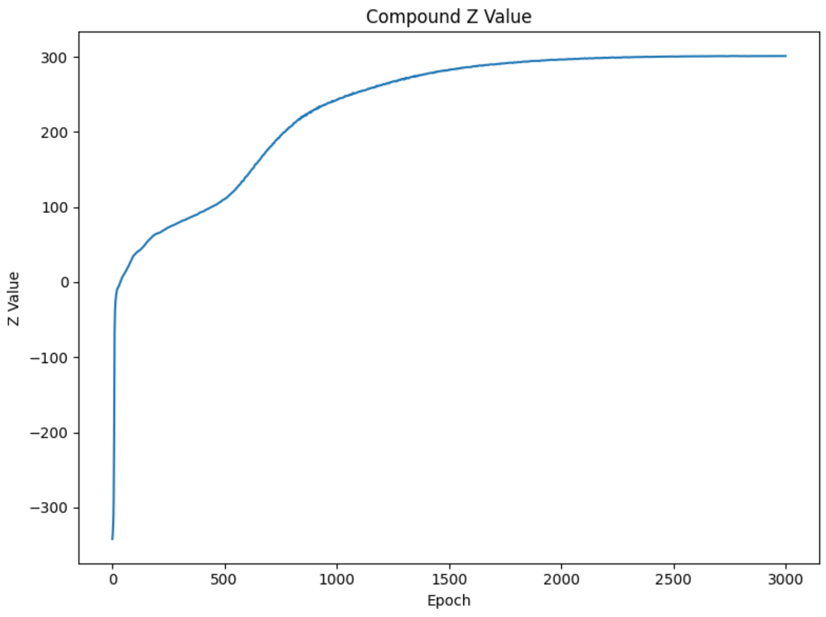


Figure 3.3. Compound Z Value

Chapter 4

Results

4.1 Overall performance on the held-out test set

Table 4.1 summarises the predictive quality of the final checkpoint (selected by the composite z-score in Section 3.8) on the *completely unseen* test-edge pool described in Section 2.2. The test split contains the same $1:10$ positive–negative ratio that approximates interactome sparsity while remaining computationally feasible; no additional negatives were generated.

Metric	Score
AUROC	0.966
AUPRC	0.885
Accuracy @ 0.50 threshold	0.967
Precision @ 0.50 threshold	0.825
Recall @ 0.50 threshold	0.809
P@500	1.000
NDCG@500	0.998

Table 4.1. Performance on the held-out test set ($n_{\text{pos}} = 5\,855$, $n_{\text{neg}} = 58\,550$).

From these statistics we can make some Key observations:

- **Global separability.** An AUROC of 0.96 indicates that in 96 % of randomly drawn *positive–negative* pairs, the model assigns the higher score to the true interaction.
- **Class-imbalance robustness.** The AUPRC of 0.89 is about 10 times the auprc of a random classifier (≈ 0.09) imposed by the $1:10$ label skew.
- **Practical hit rate.** Of the 500 highest-ranked predictions, all of them correspond to withheld positives ($P@500 = 1.000$).
- **Thresholded operation.** Using the default 0.5 cut-off yields a 82 % precision versus 81 % recall trade-off, showing strong performance also on the general task.

Overall, the model achieves great discriminative power on the STRING 950 benchmark, while delivering an experimentally actionable short-list whose error rate is below 1 %. Subsequent sections dissect the factors behind these results (precision–recall behaviour, rank-based metrics, topological biases and scalability).

4.2 Precision–recall and threshold analysis

Figure 4.1 (left) shows the precision–recall (PR) curve obtained on the *test* query set; Figure 4.2 (right) displays the corresponding histogram of predicted probabilities for positives and negatives.

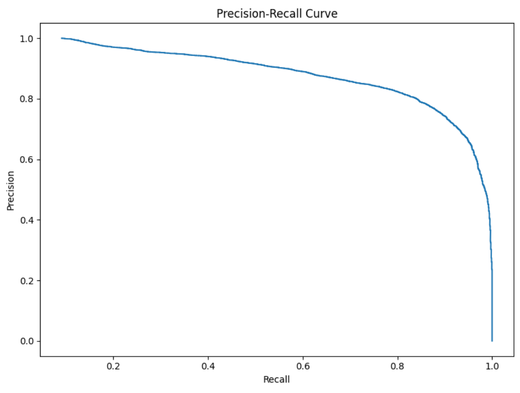


Figure 4.1. Precision–recall curve

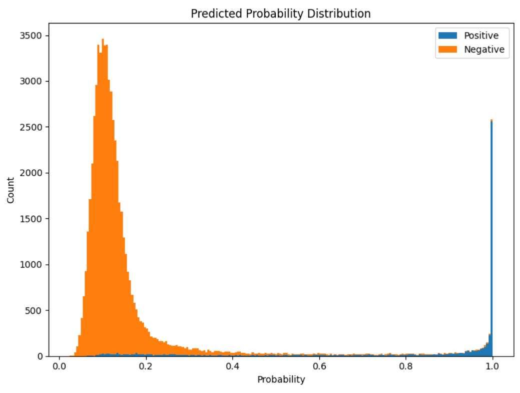


Figure 4.2. Predicted–probability histogram

4.2.1 Thresholds analysis

The PR curve is stable until recall reaches ≈ 0.83, with precision staying above 0.80 throughout that segment. The steep drop-off only begins once recall exceeds 0.90. This behaviour indicates that the model can retrieve a large fraction of true interactions before incurring a substantial penalty in precision. The histogram in Fig. 4.2 is **bimodal**: Negatives form a narrow bell centred at $p \approx 0.16$ with tail mass vanishing beyond $p = 0.35$, while Positives create a spike at the extreme right ($p \approx 0.97 - 1.00$), with negligible overlap. Because the two modes are well separated, a single global threshold can discriminate the classes without fine-tuning.

4.2.2 Operating-point selection

Accuracy plateau for thresholds in the interval $\tau \in [0.40, 0.80]$. A conventional choice $\tau = 0.50$ therefore offers a robust trade-off and is used for all subsequent analyses. At this cut-off as previously stated in Table 4.1 the test metrics are

$$\text{precision} = 0.825, \quad \text{recall} = 0.809$$

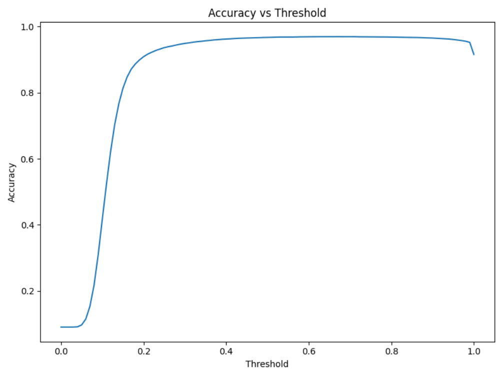


Figure 4.3. Accuracy on the varying threshold

4.2.3 Implications

High precision at moderate recall aligns with experimental priorities: wet-lab validation will focus on a few hundred strongest predictions rather than an exhaustive screen. The minimal score overlap suggests that probabilistic outputs are **well**

calibrated in the high-confidence region, an advantage when ranking candidates by expected utility. Because performance is stable across a broad threshold window, minor distributional shifts (e.g. different negative-sampling seeds) are unlikely to degrade downstream precision.

4.3 Top-N ranking metrics ($P@K$, NDCG)

Table 4.2 reports precision-at-K ($P@K$) and normalised discounted cumulative gain (NDCG@K) for several window sizes on the held-out *test* split.

	K=50	K=100	K=500	K=1000	K=3000
$P@K$	1.000	1.000	1.000	1.000	0.984
NDCG@K	1.000	1.000	1.000	1.000	0.986

Table 4.2. Top-N performance on the test set

.

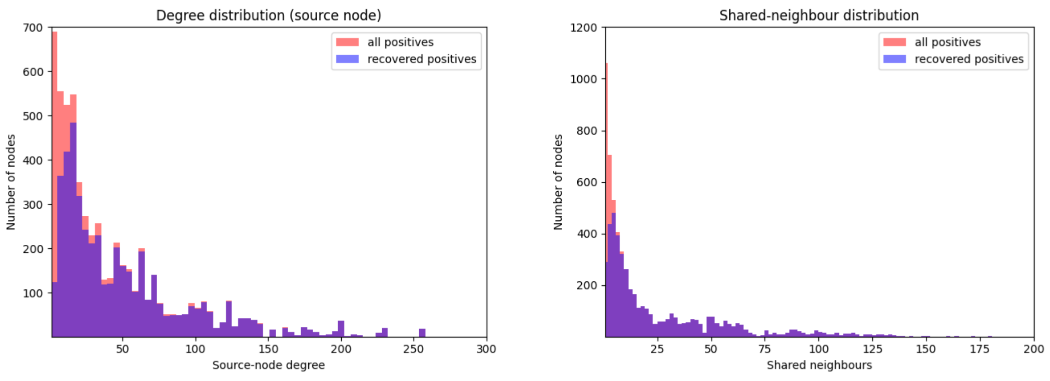
Interpretation

The model presents a perfect early ranking. All of the first **1000** predictions are ground-truth positives; $P@K = 1.0$ and NDCG@$K = 1.0$ indicate flawless ordering within that window. Precision only begins to drop when the window widens to 3 000 candidates, yet remains exceptionally high ($P@3000 = 0.984$). The corresponding NDCG@3000 of 0.986 shows that the few remaining false positives are pushed towards the bottom of the list.

4.4 Topological-bias analysis

Message-passing GNNs often favour edges that connect well-embedded, information-rich vertices. To quantify this tendency the test positives were partitioned into *recovered* (true–positive, TP) and *missed* (false-negative, FN) sets and compared against the complete positive pool along two elementary graph features: node degrees and shared neighbours.

Figures 4.4a and 4.4b plot the corresponding histograms for *all* positives (red) and *recovered* positives (purple). A numerical summary is given in Table 4.3.



(a) Source-node degree **(b)** Shared-neighbour count

Figure 4.4. Topology of recovered versus all positives (test split, $\tau = 0.5$).

Subset	Mean	Median	95 %ile
Source-node degree			
True positives (TP)	51.85	35	143
False positives (FP)	30.92	17	103
False negatives (FN)	11.98	5	44
True negatives (TN)	10.70	5	44
All positives (TP + FN)	44.07	26	140
Target-node degree			
TP	55.53	36	183
FP	32.85	17	118
FN	11.94	5	48
TN	10.86	5	45
TP + FN	46.89	27	172
Shared neighbours			
TP	27.51	14	99
FP	0.91	0	5
FN	1.18	1	4
TN	0.01	0	0
TP + FN	23.13	9	91

Table 4.3. Degree and shared-neighbour statistics for each prediction category. TP = true positives, FP = false positives, FN = false negatives, TN = true negatives. "TP + FN" corresponds to the entire positive class.

4.4.1 Observed bias

- **Degree.** Recovered edges originate, on average, from nodes that are ~18 % more connected than the positive baseline (mean 52 vs. 44). The difference is most pronounced in the lower half of the distribution: the blue histogram in Fig. 4.4a is strongly depleted for $k < 10$.
- **Shared neighbours.** A similar shift is evident in Fig. 4.4b: median shared-neighbour count rises from 9 to 14, and the heavy tail beyond 30 is basically completely recovered.

4.4.2 Failure mode

False-negative edges cluster in the complementary region—both end-points have low degree (mean of $\bar{k} = 12$) and share at most a handful of common neighbours. Because information can only propagate along existing paths, embeddings of such peripheral nodes remain weakly informed, and the pair decoder assigns them sub-threshold scores.

4.4.3 Implications

The model is highly effective inside dense modules of the interactome but under-performs on sparsely connected proteins. While this bias is acceptable for applications that prioritise high-confidence hits in well-studied regions, recovering interactions on the graph periphery would require either (i) augmenting low-degree nodes with external features (e.g. structural motifs, expression profiles) or (ii) adopting architectures with longer effective receptive fields or (iii) use a heavier more connected graph.

Chapter 5

Case Study: NOTCH2 Interaction Landscape

5.1 NOTCH2 overview

NOTCH2[25] is a transmembrane receptor that belongs to the canonical *Notch* signalling family (NOTCH1–4) [26]. From a graph-analytics perspective the protein exhibits the following properties in STRING v12:

- **Identifier.** STRING ID `9606.ENSP00000256646`.
- **High-confidence degree.** 14 neighbours at `combined_score`$\geq$ 950 (the threshold used for training).
- **Full-graph degree.** 2 294 neighbours in non filtred graph.
- **Functional classes (GO slim).** Signal transduction, transcriptional regulation, and calcium ion binding .

5.2 Data and experimental design

The case study was conducted on 2 different expressions of the STRING interactome

- **LC-250 Graph**: A low confidence filtering of the STRING interactome, removing edges with `combined_score`$<$ 250 from the message passing graph. This results in a graph with 19 598 unique proteins and about 4,8M edges.
- **HC-950 Graph**: The same graph used for the training, validation and testing of the model, with 10 430 unique proteins and 0,12M edges.

Candidate–edge pools

For each graph configuration a separate set of query edges is generated; the construction rules ensure that evaluation goals differ between the sparse, high-confidence graph and the dense whole-proteome graph.

LC-250 graph ($m = 4,8$ M edges): Start from the complete STRING v12 interactome and retain every edge with `combined_score` ≥ 250. For the query protein (NOTCH2) enumerate the $19\,598 - 1 = 19\,597$ possible partner proteins. Remove the 781 partners that already have an edge at ≥ 250; the remaining $\sim 16\,700$ pairs constitute the **candidate pool**. When evaluating, label a candidate as "recovered" if a sub-250 edge exists in the raw STRING file and "novel" if no edge of any score is present. This lets us measure if the model can somewhat recover even low `combined score` edges and if they are scored in a statistically significant way compared to novel edges

HC-950 graph ($m = 0.12$ M edges): Cache all NOTCH2 edges from the unfiltered interactome (scores $1 - 999$); these will be excluded later. Build the high-confidence graph by keeping only edges with `score` ≥ 950. NOTCH2 now has 14 neighbours. Form candidate pairs between NOTCH2 and every other protein **except** the cached partners from step HC1. The pool therefore contains only edges that are *completely absent* from STRING v12, so only potentially novel edges.

The LC-250 pool lets us test edge *recovery* on a dense graph and novel prediction on a different graph, whereas the HC-950 pool isolates the model's ability to recommend entirely novel interactions on the same graph of the training.

5.3 Quantitative results (LC-250 graph)

5.3.1 Score distribution: recovered vs. novel edges

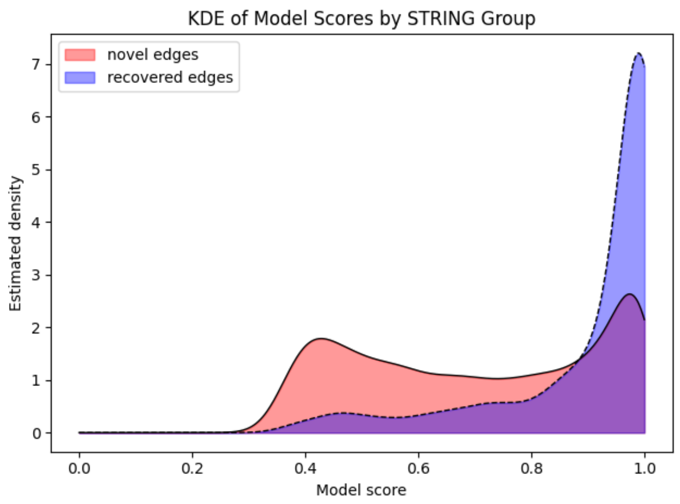


Figure 5.1. Kernel-density estimate of model scores for NOTCH2 candidate edges on the LC-250 graph. "Recovered" = STRING edge with $0 < \texttt{score} < 250$; "Novel" = no STRING evidence.

Figure 5.1 compares the model's probabilistic output for the two label groups available in the low-confidence graph:

- **Recovered edges** (blue) are links that exist in STRING but were hidden during training because their confidence score is below 950. The density forms a sharp spike at $p \approx 0.95$, indicating that the network correctly *rescues* the vast majority of these interactions despite never having seen them before and being in general very low confidence.

- **Novel edges** (red) have `score` $= 0$ in STRING v12. Their distribution is bimodal: a broad shoulder centred around $p \approx 0.45$ and a pronounced secondary peak that overlaps the recovered cluster at $p \approx 0.9$.

Implications: The right-hand overlap demonstrates that many high-scoring predictions are entirely *novel* to STRING, not merely re-discoveries of low-confidence evidence. The left-hand shoulder shows that the model is conservative for edges with weak topological support; their probabilities remain below the 0.5 operating threshold chosen in Section 4.2.

5.4 Quantitative results (HC-950 graph)

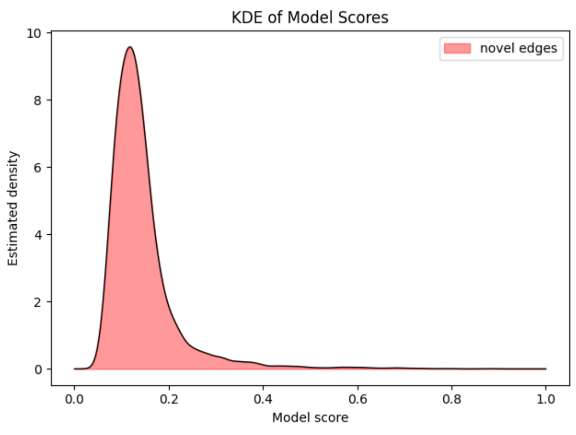


Figure 5.2. Kernel-density estimate of model scores for the HC-950 graph. By construction every candidate edge is novel (absent from STRING).

The second inference was run on the sparse HC-950 graph, after removing every protein that STRING lists as an existing NOTCH2 partner (even at very low confidence) as described in section 5.2. The resulting candidate pool contains only edges that are completely absent from STRING; none can be labelled "recovered."

Figure 5.2 shows the score kernel density: The high-probability spike observed in the LC-250 experiment (Section 5.3.1) disappears; the distribution peaks at $p \approx 0.12$ and falls off quickly above $p = 0.25$. The maximum score in this run is $p = 0.88$; no edge reaches the near-certainty region ($p \geq 0.95$) that was common when lower-confidence neighbours were present.

5.5 Interpretation: graph density, topology cues, and model behaviour

The two inference runs make it unambiguous that the encoder–decoder is highly sensitive to the **topological fabric** in which the query protein is embedded. On the dense LC-250 interactome, NOTCH2 sits in a rich neighbourhood: 781 first-hop partners fill the gap between the 14 high-confidence edges retained in the HC-950 graph and the 2 294 edges present in the unfiltered STRING file. Each of those additional vertices introduces new two-step paths and inflates the count of shared neighbours, exactly the kind of information that TAGConv and the attention layer transmit. Once these multi-hop signals are aggregated, the decoder is willing to emit probabilities close to one—even for edges that STRING currently scores below 250 or has not registered at all. In other words, the model behaves "aggressively" because the dense graph supplies a strong signal: nodes that co-occupy a well-connected cluster or share many friends-of-friends with NOTCH2 are treated as highly plausible interactors.

Switching to the sparse HC-950 graph to which the model is more accustomed removes virtually all of those weak or indirect cues. NOTCH2 now has only 14 neighbours, and those neighbours themselves live in a reduced network. The message-passing pipeline therefore propagates far less information. The decoder reacts by lowering its confidence: the score density peaks around 0.16 and barely any edge crosses the 0.9 mark. The same architecture thus oscillates between "optimistic" and "cautious" modes purely in response to graph density, illustrating a fundamental bias that accompanies neighbourhood-based GNNs.

Although this behaviour does not constitute classical over-fitting — the model still generalises to held-out edges within the training graph — it does mean that the scoring function is *calibrated* for the topological statistics of the data it saw during optimisation. Feeding it a graph that is almost two orders of magnitude denser pushes those statistics far outside the training regime; the heuristics encoded in the learned filters are "blown out of scale", and every high-degree shortcut is interpreted as a strong positive signal.

The practical challenge becomes visible as soon as one attempts richer interactome inference. Message passing on the 4.8-million-edge graph of LC-250 already exhausts 140 GB of H200 memory. Sparse mini-batch techniques such as neighbour sampling, using batches of local clusters if entire graph, or GraphSAGE–style inductive passes would alleviate the memory wall, but at the cost of discarding exactly part of the information preset in the graph. A trade-off is inevitable: one can preserve accuracy by working on the entire graph with expensive hardware, or can accept a controlled loss of information in exchange for commodity-scale resources.

5.6 Top-ranked NOTCH2 partners and biological plausibility

The quantitative metrics in Section 5.3 confirm that the encoder–decoder is a strong ranker, but numbers alone do not establish biological value. To gauge plausibility, the ten highest-scoring edges from each inference run were examined manually in literature. The discussion below is deliberately qualitative: the aim is to show why a computer-science model might highlight a protein, not to claim definitive biochemical proof.

Large graph (STRING ≥ 250)

The top five novel predictions (edges where STRING score = 0, meaning truly no prior record, and model score = 1.0) were PLEKHA4, CALML4, AKIRIN2, CALML6, and LRRK2. It was noteworthy that the model assigned a score of 1.0 to all of these candidatesl. Upon literature review, some of these have some hints of connection to NOTCH biology, while others are pretty much unknown in this context.

LRRK2. This prediction is probably the one with the strongest literature support: LRRK2 has been reported to modulate the Notch pathway. Specifically, LRRK2 (along with partners like HERC2 and NEURL4) can affect Notch signal strength [27] . This connection was shown in the context of neuronal cells and linked to Parkinson's pathology. So, even though STRING didn't list a direct interaction for NOTCH2–LRRK2, the model somehow surfaced LRRK2 as a top candidate – and in this case, the biological rationale exists in literature.

PLEKHA4. No direct interaction with NOTCH2 is known, but interestingly PLEKHA4 is involved in Wnt signalling regulation and cell polarity. In a bioinformatics study of breast cancer, low PLEKHA4 expression was associated with up-regulation of the Notch pathway [28], hinting at a relationship. This suggests PLEKHA4 might indirectly modulate pathways that include Notch.

CALML4 & CALML6. There's no literature linking CALML4/6 to Notch directly. However, it's worth noting that Notch receptor activity can be calcium/calmodulin-dependent. For example, one study showed connection between calcium/calmodulin and NOTCH1 (not NOTCH2) signalling [29] . It's conceivable that CALML4/6could influence Notch2 signalling or localization, but as of now, this is purely hypothetical – they remain novel predictions without supporting evidence and could be false positives.

AKIRIN2. There's no direct tie between AKIRIN2 and NOTCH2 that could be found.

High-confidence graph (STRING ≥ 950)

Here the model's top predictions were more conservative. The highest-scoring NOTCH2 partner genes (the only ones with model scores >0.8 in this run) were JUP, FBN1, and PATJ.

PATJ. Patj is one of the organiser proteins that keep neighbouring cells together. The model gave PATJ a fairly high score for partnering with NOTCH2, and that lines up with a recent finding: In brain endothelial cells, loss of PATJ triggered changes in gene expression that included downregulation of Notch pathway activity [30]. So among the "novel" predictions, PATJ actually has some experimental support, not as a direct binding partner proven by biochemistry, but as a modulator of Notch signaling outcomes. No one has shown PATJ linking with NOTCH2 directly, but the functional link is there, so this prediction could be believable and worth testing.

JUP (plakoglobin). At first glance, JUP has no known direct interaction with Notch2. But it can make sense in terms of cellular context: Notch receptors function at the cell membrane where cell-cell contacts occur, and proteins like JUP are found in those same junctional areas. For instance, NOTCH2 signaling often intersects with Wnt pathways, while JUP can antagonize canonical Wnt signaling in the nucleus. This suggests a pathway crosstalk context, but no literature to date demonstrates a direct NOTCH2JUP interaction or shared complex of any kind.

FBN1 (fibrillin-1). Fibrillin-1 is a large structural protein that helps give tissues their elasticity, and Is heavily connected with marfan syndrome . At first glance it seems unrelated to a signalling receptor like NOTCH2, but some studies show that proteins in the surrounding matrix can attach to Notch receptors or their helpers and influence the signal [31]. So, even though nobody has shown FBN1 binding NOTCH2 directly, the idea that fibrillin could interact with NOTCH2 is feasible.

What all this means

- On a dense graph the model is bold: it hands out perfect scores to proteins like LRRK2 that later turn out to have published Notch links, but also to less obvious candidates such as CALML6.
- On a sparse graph it is cautious; only a handful of edges break 0.8. Those few, however, still make biological sense (PATJ, FBN1).
- The exercise shows both sides of topology-driven GNNs: they can rediscover hidden biology, yet their confidence depends heavily on how rich the surrounding network is.

For anyone planning wet-lab follow-up, the takeaway is simple: use the model to narrow the search space, then let domain experts decide which of the high-scoring novelties deserve bench time.

Chapter 6

Conclusion and Future Work

6.1 Summary of Contributions

This thesis introduced a hardware–aware graph-neural-network (GNN) pipeline that fuses **ProtT5** sequence embeddings with topology from **STRING v12** to predict missing protein–protein interactions (PPIs). Collectively, the results demonstrate that an interpretable, resource-efficient GNN can deliver near-perfect early-precision on large, noisy biological networks and generate meaningful hypotheses for molecular biology.

6.2 Limitations

Despite its strengths, the present work has several constraints:

- **Topology bias**. Performance degrades for edges involving low-degree or peripheral proteins; high confidence is obtained mainly in dense network regions.
- **STRING specificity**. Training and evaluation rely on a single database and a fixed confidence cutoff, limiting external generalisability.
- **No baseline comparison**. Although internal metrics are strong, direct benchmarking against heuristic or other gnn-based methods are very challenging considering the different resources, datasets and scopes of the projects.
- **Scalability**. Full-graph inference above five million edges exceeded GPU memory; neighbourhood-sampling was not explored.

Understanding and mitigating these limitations motivates the following research directions.

6.3 Future Work: Making the Model Truly Versatile

6.3.1 Cross-Database Generalisation

1. **Multi-source training.** Merge BioGRID, IntAct and multiple STRING confidence tiers into a unified, hetero-confidence graph. Train with reliability-aware losses that weight edges by source quality, forcing the encoder to cope with variable evidence density.

2. **Zero-shot transfer tests.** Hold out one database (e.g. IntAct) during training and evaluate on its edges only. A marked drop would reveal over-fitting to STRING-specific wiring patterns; recovery strategies include domain-adversarial training or graph mixup across sources.

6.3.2 Explainability and Feature Importance

1. **Feature-masking attribution.** Iteratively mask ProtT5 dimensions, node degrees, or shared-neighbour counts while monitoring the drop in logit for a given edge. Ranking the most disruptive masks yields an importance spectrum for **sequence** versus **topology**.
2. **GNNExplainer / PGExplainer.** Apply node- and subgraph-level explanation tools to visualise which hops or motifs dominate a prediction, pinpointing potential over-reliance on hubs.

6.3.3 Ablation and Robustness Studies

1. **Encoder component ablations.** Train variants that drop TAGConv, TransformerConv or GIN layers to quantify each block's marginal gain in performance.
2. **Topology-only vs. Features-only.** Compare (i) shuffled ProtT5 vectors and (ii) identity-feature nodes to isolate the added value of language-model embeddings.

6.3.4 Scalable Training Dynamics

1. **Subgraph batching.** Integrate Cluster-GCN or GraphSAINT so that 4–5M-edge graphs (e.g. STRING$\geq$ 250) fit into 16GB GPUs without sacrificing context.
2. **Variations on graph size.** Start training on the 950-graph, progressively inject 700-, 500-, 250-score edges, and finally the full STRING graph. This staged exposure teaches the encoder to remain calibrated across sparsity regimes.

6.4 Concluding Remarks

The present work shows that a carefully regularised, interpretable GNN can achieve near-perfect early-precision on a high-confidence interactome while remaining GPU-affordable. Making the model *truly versatile* now hinges on three pillars:

- **Generalisation**: learn from—and be evaluated on—multiple PPI sources of differing density and reliability.
- **Explainability**: expose which sequence features, paths or local motifs drive a prediction, enabling trust and biological insight.
- **Scalability**: handle million-edge graphs via subgraph batching and curriculum schedules without degrading calibration.

Pursuing these directions will convert the current high-precision prototype into a robust, cross-database recommender that accelerates the discovery of genuine yet previously overlooked protein interactions.

Bibliography

[1] Luck K, Sheynkman GM, Zhang I, Vidal M. "Proteome-Scale Human Interactomics." *Trends Biochem Sci.* 2017;42(5):342-354. doi:10.1016/j.tibs.2017.02.006

[2] Schaefer M.H., Lopes T.J.S., Mah N. *et al.* "Adding protein context to the human protein–protein interaction network to reveal meaningful interactions." *PLoS Computational Biology*, 9(1):e1002860, 2013. https://doi.org/10.1371/journal.pcbi.1002860

[3] Wang X.-W., Madeddu L., Spirohn K. *et al.* "Assessment of community efforts to advance network-based prediction of protein–protein interactions." *Nature Communications*, 14:1582, 2023. https://doi.org/10.1038/s41467-023-37079-7

[4] Zhang X.-M., Liang L., Liu L. *et al.* "Graph Neural Networks and Their Current Applications in Bioinformatics." *Frontiers in Genetics*, 12:690049, 2021. https://doi.org/10.3389/fgene.2021.690049

[5] Li M.M., Huang K., Zitnik M. "Graph representation learning in biomedicine and healthcare." *Nature Biomedical Engineering*, 6(12):1353–1369, 2022. https://doi.org/10.1038/s41551-022-00942-x

[6] Szklarczyk D. *et al.* "The STRING database in 2025: protein networks with directionality of regulation." *Nucleic Acids Research*, 53(D1):D730–D737, 2025. https://doi.org/10.1093/nar/gkae11135

[7] Oughtred R. *et al.* "The BioGRID database: a comprehensive biomedical resource of curated protein, genetic, and chemical interactions." *Protein Science*, 30(1):187–200, 2021. https://doi.org/10.1002/pro.3978

[8] Orchard S. *et al.* "The MIntAct project: IntAct as a common curation platform for 11 molecular interaction databases." *Nucleic Acids Research*, 42(Database Issue):D358–D363, 2014. https://doi.org/10.1093/nar/gkt1115

[9] Elnaggar A., Heinzinger M., Dallago C. *et al.* "ProtTrans: toward understanding the language of life through self-supervised learning." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(10):7112–7127, 2021. https://doi.org/10.1109/TPAMI.2021.3095381

[10] Hu D., Szklarczyk D., von Mering C., Jensen L. J. "SPACE: STRING proteins as complementary embeddings." *bioRxiv*, 2024. https://doi.org/10.1101/2024.11.25.625140

[11] Kipf T. N. and Welling M. "Semi-supervised classification with Graph Convolutional Networks." In *Proc. ICLR*, 2017. https://doi.org/10.48550/arXiv.1609.02907

[12] torch_geometric.nn.conv.GCNConv documentation. https://pytorch-geometric.readthedocs.io/en/stable/generated/torch_geometric.nn.conv.GCNConv.html .

[13] Morris C., Ritzert M., Fey M. *et al.* "Weisfeiler and Leman go neural: higher-order graph neural networks." In *Proc. AAAI*, 2019. https://doi.org/10.48550/arXiv.1810.02244

[14] torch_geometric.nn.conv.GraphConv documentation. https://pytorch-geometric.readthedocs.io/en/stable/generated/torch_geometric.nn.conv.GraphConv.html .

[15] Veličković P., Cucurull G., Casanova A. *et al.* "Graph Attention Networks." In *Proc. ICLR*, 2018. https://doi.org/10.48550/arXiv.1710.10903

[16] torch_geometric.nn.conv.GATConv documentation. https://pytorch-geometric.readthedocs.io/en/latest/generated/torch_geometric.nn.conv.GATConv.html .

[17] Du J., Zhang S., Wu G., Moura J. M. F., Kar S. "Topology adaptive graph convolutional networks." *arXiv*:1710.10370, 2017. https://doi.org/10.48550/arXiv.1710.10370

[18] torch_geometric.nn.conv.TAGConv documentation. https://pytorch-geometric.readthedocs.io/en/stable/generated/torch_geometric.nn.conv.TAGConv.html (accessed 15 Jun 2025).

[19] Shi Y., Huang Z., Feng S. *et al.* "Masked Label Prediction: unified message passing model for semi-supervised classification." In *Proc. ICLR*, 2021.

[20] Vaswani A., Shazeer N., Parmar N. *et al.* "Attention is all you need." In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[21] torch_geometric.nn.conv.TransformerConv documentation (v 2.5.2). https://pytorch-geometric.readthedocs.io/en/stable/generated/torch_geometric.nn.conv.TransformerConv.html .

[22] Xu K., Hu W., Leskovec J., Jegelka S. "How powerful are Graph Neural Networks?" In *Proc. ICLR*, 2019.

[23] torch_geometric.nn.conv.GINConv documentation (v 2.5.2). https://pytorch-geometric.readthedocs.io/en/stable/generated/torch_geometric.nn.conv.GINConv.html .

[24] torch_geometric.transforms.RandomLinkSplit documentation (v 2.5.2). https://pytorch-geometric.readthedocs.io/en/stable/generated/torch_geometric.transforms.RandomLinkSplit.html.

[25] https://string-db.org/network/9606.ENSP00000256646

[26] Artavanis-Tsakonas S., Rand M.D., Lake R.J. "Notch signaling: cell fate control and signal integration in development." *Science*, 284(5415):770–776, 1999. https://doi.org/10.1126/science.284.5415.770

[27] Imai Y. *et al.* "The Parkinson's disease-associated protein kinase LRRK2 modulates Notch signalling through the endosomal pathway." *PLOS Genetics*, 11(9):e1005503, 2015. https://doi.org/10.1371/journal.pgen.1005503

[28] Li L., Feng Z., Zhao Y. *et al.* "Prognostic value of PLEKHA4 and its correlation with tumour-infiltrating immune cells in breast cancer: a comprehensive study based on bioinformatics and clinical analysis validation." *Translational Cancer Research*, 13, 2023. https://doi.org/10.21037/tcr-24-67

[29] Mamaeva O. A. *et al.* "Calcium/calmodulin-dependent kinase II regulates Notch-1 signalling in prostate cancer cells." *Journal of Cellular Biochemistry*, 106(1):25–32, 2009. https://doi.org/10.1002/jcb.21973

[30] Medina-Dols A., Cañellas G., Capó T. *et al.* "Role of PATJ in stroke prognosis by modulating endothelial-to-mesenchymal transition through the Hippo/Notch/PI3K axis." *Cell Death Discovery*, 10:85, 2024. https://doi.org/10.1038/s41420-024-01857-z

[31] Jespersen K., Liu Z., Li C. *et al.* "Enhanced Notch3 signalling contributes to pulmonary emphysema in a murine model of Marfan syndrome." *Scientific Reports*, 10:10949, 2020. https://doi.org/10.1038/s41598-020-67941-3