SUBGRAPH REPRESENTATION LEARNING ON HETEROGENEOUS GRAPHS

Anonymous authors
Paper under double-blind review

ABSTRACT

Subgraph representation learning on heterogeneous graphs requires accounting jointly for subgraph membership and the types of nodes and relations in a shared base graph. Existing approaches primarily address node- or graph-level heterogeneity, leaving subgraph-level challenges less explored. We propose Cross-Type Anchor Projection (CTAP), which decomposes the target subgraphs and the base graph by node type, representing each type-specific node set within its surrounding graph. Cross-type projection maps each other-type node (an anchor) to the subgraph of its neighbors, yielding views that relate these sets to anchors and capture long-range dependencies. Semantic fusion combines representations learned on the views, while membership pooling incorporates cross-type neighbor and constituent-node information before the final fusion across node types. Among the compared methods, CTAP achieves the highest mean test micro-F1 on five of six new datasets drawn from team discovery, rare-disease diagnosis, and gene/protein set analysis.

1 INTRODUCTION

Subgraph representation learning aims to represent an object defined by a subset of nodes within a shared base graph. Such objects include a patient's observed phenotypes, a group of proteins participating in a biological process, and a team of collaborating individuals. Their properties depend on both the nodes they contain and their relationships with the surrounding graph. SubGNN (Alsentzer et al., 2020) formulated this setting for subgraphs of varying sizes and connectivity patterns, and later methods (Wang & Zhang, 2022; Jacob et al., 2023; Zhong et al., 2025; Kim & Oh, 2024; Wu et al., 2025; Kim & Oh, 2025) refined how membership, position, and dependencies between subgraphs are encoded in homogeneous base graphs.

For these objects, however, the base graphs are heterogeneous, linking phenotypes to genes and diseases in rare-disease diagnosis (Alsentzer et al., 2025), proteins to pathways and processes in gene/protein set analysis (Sharma et al., 2022; Chang, 2012; Levine et al., 2006), and experts to skills and contextual entities in team discovery (Hamidi Rad et al., 2023). Heterogeneity is typically handled either at the node level by heterogeneous graph neural networks (HGNNs) (Yang et al., 2023; Lv et al., 2021) or at the whole-graph level by pooling methods such as HINPool (Hong et al., 2026). Application-specific approaches likewise rely on node-level mechanisms (Hamidi Rad et al., 2023; Alsentzer et al., 2025), so challenges that arise at the subgraph level remain less explored.

We identify three such challenges in subgraph representation learning on heterogeneous graphs. First, nodes of different types within a target may convey distinct aspects of the same object, so each type-specific node set must be represented separately within its surrounding graph context. Second, a node of another type may connect to only part of a type-specific set, so their relation must be captured at the set level. Third, intervening nodes of other types create indirect connections among members of a type-specific set, requiring long-range dependency modeling. Appendix B illustrates these challenges.

We propose Cross-Type Anchor Projection (CTAP) to address these challenges. For the first challenge, CTAP decomposes both the target subgraphs and the base graph into type-specific subproblems. Within each subproblem, cross-type projection represents each node of another type, called an anchor, as the subgraph induced by its neighbors in that subproblem. The resulting node-overlap and edge-connectivity views relate each type-specific set to anchors and model long-range dependencies,

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

addressing the second and third challenges. Same-type views additionally capture information from neighbors of the same type. CTAP applies message passing to each view and combines the outputs using semantic fusion. Membership pooling then updates node representations with cross-type neighbor information and subgraph representations with constituent-node information. After repeating these updates across layers, CTAP fuses the type-specific representations of each target into its final representation.

We construct six datasets for subgraph representation learning on heterogeneous graphs, covering team discovery, rare disease prediction, and biological function prediction. CTAP achieves the highest mean test micro-F1 on five of the six datasets among the compared methods. Component ablations further evaluate the contributions of its views and membership pooling.

2 Related Work

Subgraph representation learning Early studies explored embedding- and pattern-based approaches to subgraph representation, including Sub2Vec (Adhikari et al., 2018) and Subgraph Pattern Neural Networks (Meng et al., 2018). SubGNN (Alsentzer et al., 2020) subsequently formulated general representation learning for subgraphs of varying sizes and connectivity patterns within a shared base graph. Later work includes the node-level GNN baseline GNNplain (Wang & Zhang, 2022) and methods such as GLASS (Wang & Zhang, 2022), PADEL (Liu et al., 2023), GPEN (Wu et al., 2025), SSNP (Jacob et al., 2023), and ISNN (Zhong et al., 2025), which incorporate subgraph membership, position, neighborhood, or dependencies between subgraphs into representation learning. S2N (Kim & Oh, 2024) translates each subgraph into a node of a new graph, and WLKS (Kim & Oh, 2025) generalizes Weisfeiler–Lehman kernels to subgraphs. GPEN, ISNN, and implicit GNNs such as EIGNN (Liu et al., 2021) further capture long-range dependencies, but these methods are designed for homogeneous base graphs. CTAP applies S2N translation to target and projected subgraphs within each type-specific subproblem.

Subgraphs on heterogeneous graphs Heterogeneous base graphs arise in applications such as team discovery, where teams are represented by experts, skills, and contextual entities, which is formalized by Hamidi Rad et al. (2023), and rare-disease diagnosis, where SHEPHERD represents patients by phenotype subgraphs in a biomedical knowledge graph (Alsentzer et al., 2025). In molecular biology, gene and protein sets are used for protein-complex function prediction (Sharma et al., 2022), gene-set regulation and tissue specificity analysis (Chang, 2012), and pathway activity analysis across tissues (Levine et al., 2006).

Graph neural networks and pooling for heterogeneous graphs HGNNs broadly fall into metapath-based and metapath-free methods. Metapath-based HGNNs, such as HAN (Wang et al., 2019), MAGNN (Fu et al., 2020), SeHGNN (Yang et al., 2023), and HCMGNN (Zhang et al., 2024), aggregate information along predefined metapaths and fuse the resulting representations; CTAP applies such semantic fusion to its views instead of metapath neighborhood graphs. Metapath-free HGNNs, such as HGT (Hu et al., 2020), Simple-HGN (Lv et al., 2021), and HHGT (Zhu et al., 2025), aggregate local messages using node and edge types. For graph-level prediction, domain-specific methods, such as UaG (Wu et al., 2021) for user modeling and MPMol (Ji et al., 2023) for molecular graphs, pool information separately by node or relation type and then combine the resulting representations, and HINPool (Hong et al., 2026) generalizes this approach. Unlike an entire graph, a target subgraph is defined within a shared base graph, so CTAP decomposes the base graph as well as the targets by type (Appendix B).

3 Problem Setting

Let $G = (\mathcal{V}, \mathcal{E}, \mathbf{X})$ denote an undirected graph with a finite node set $\mathcal{V} = \{v_1, v_2, \ldots, v_{|\mathcal{V}|}\}$, an edge set $\mathcal{E} \subseteq \{\{u, v\} \mid u, v \in \mathcal{V}\}$, and a node feature matrix $\mathbf{X} \in \mathbb{R}^{|\mathcal{V}| \times d}$, whose i-th row $\mathbf{X}_i$ is the feature vector of v_i; we use the same d for hidden representations for simplicity. Each edge $\{u, v\}$ is unordered and counted once, including a self-loop when $u = v$. The adjacency matrix $\mathbf{A}$ of G has entry $\mathbf{A}_{uv}$ equal to the weight of $\{u, v\}$ if the edge exists and zero otherwise, with unit weights for unweighted graphs. A heterogeneous graph additionally assigns each node v a type $f_{\mathcal{V}}(v) \in \mathcal{T}_{\mathcal{V}}$ and each edge $\{u, v\}$ a type $f_{\mathcal{E}}(\{u, v\}) \in \mathcal{T}_{\mathcal{E}}$. We assume that each edge type connects a fixed pair of node types, which holds for all our datasets; several edge types may connect the same pair. A

subgraph of G is denoted by $S = (\mathcal{V}_S, \mathcal{E}_S, \mathbf{X}_S)$, where $\mathcal{V}_S \subseteq \mathcal{V}$, $\mathcal{E}_S \subseteq \{\{u, v\} \in \mathcal{E} \mid u, v \in \mathcal{V}_S\}$, and $\mathbf{X}_S \in \mathbb{R}^{|\mathcal{V}_S| \times d}$ contains the rows of $\mathbf{X}$ corresponding to $\mathcal{V}_S$, in the base graph's node order. When $\mathcal{E}_S = \{\{u, v\} \in \mathcal{E} \mid u, v \in \mathcal{V}_S\}$, S is the subgraph of G induced by $\mathcal{V}_S$. Given a base graph G and a collection $\mathcal{S} = \{S_1, S_2, \ldots, S_n\}$ of target subgraphs, or targets, the goal of subgraph representation learning is to learn a representation $\mathbf{h}_{S_i} \in \mathbb{R}^{d'}$ for each S_i. Subgraph representation learning is commonly formulated in the node-induced setting: node sets $\{\mathcal{V}_{S_1}, \ldots, \mathcal{V}_{S_n}\}$ are given, and representations are learned for the subgraphs they induce in G. We adopt this setting in its transductive form, as in prior work (Alsentzer et al., 2020; Wang & Zhang, 2022): all targets lie in one shared base graph, and the model is trained with the labels of training targets only. We evaluate the representations on subgraph classification, where a classifier predicts the label vector $\mathbf{y}_{S_i} \in \{0, 1\}^{d_y}$ of each S_i over d_y classes from $\mathbf{h}_{S_i}$; $\mathbf{y}_{S_i}$ is one-hot for single-label and multi-hot for multi-label tasks. Appendix A summarizes the notation.

4 Type-Aware Decomposition and Views

4.1 Decomposition and Projection

Because a target subgraph is represented relative to the shared base graph, we decompose both the targets and the base graph by node type. For each node type τ, we form a subproblem whose base graph G^τ and target subgraphs S_i^τ are induced by the type-τ nodes of G and S_i, respectively, with node sets $\mathcal{V}^\tau$ and $\mathcal{V}_{S_i}^\tau$. Targets containing no type-τ nodes are excluded; the remaining targets form $\mathcal{S}^\tau$. Each subproblem is itself a subgraph representation learning problem.

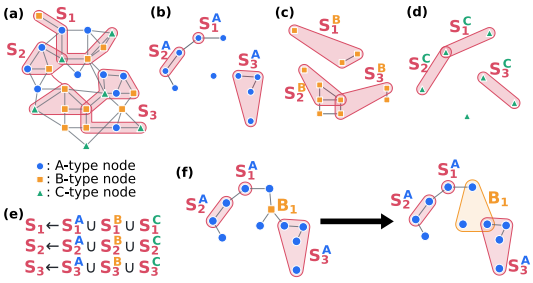

Figure 1: Type-aware decomposition and cross-type projection.

Figure 1 illustrates type-aware decomposition and cross-type projection. Panels (a)–(d) show targets S_1, S_2, S_3 in G and their type-specific parts in G^A, G^B, G^C; each base graph includes the nodes and edges outside the marked target regions. Panel (e) shows that these parts partition the nodes of each target, whose representation combines theirs.

Nodes of other types connected to type-τ nodes still provide useful information for the subproblem. Rather than adding them to G^τ, we represent each such node by the set of its type-τ neighbors, separately for each edge type. We assume that each edge type connects a fixed pair of node types, as in all our datasets; several edge types may connect the same pair. Formally, for the subproblem of node type τ, consider a node $q \in \mathcal{V}^\kappa$ of type $\kappa \neq \tau$ connected to nodes of type τ through edges of type $r \in \mathcal{T}_\mathcal{E}$. Let $\mathcal{V}_q^{\tau,r} = \{v \in \mathcal{V}^\tau \mid \{v, q\} \in \mathcal{E},\ f_\mathcal{E}(\{v, q\}) = r\}$. We define the projection of node q through edge type r into the τ-type subproblem as the subgraph $P_q^{\tau,r}$ induced by $\mathcal{V}_q^{\tau,r}$ in G^τ, and call q an anchor node. In panel (f), the B-type node B_1 on the left is represented by the subgraph induced by its A-type neighbors on the right.

4.2 Views

Within the τ-type subproblem, we relate targets and anchors through Subgraph-To-Node (S2N) translation (Kim & Oh, 2024), which maps each subgraph to a node and each pair of subgraphs S, S' to an edge weight $T_e(S, S')$. Let $\mathcal{C}^\tau$ be the set of cross-type relations incident to node type τ:

$$\mathcal{C}^\tau = \{f_\mathcal{E}(\{v, q\}) \mid \{v, q\} \in \mathcal{E},\ f_\mathcal{V}(v) = \tau,\ f_\mathcal{V}(q) \neq \tau\}. \tag{1}$$

For each $r \in \mathcal{C}^\tau$, let $\mathcal{Q}^{\tau,r}$ contain the anchors q with nonempty $\mathcal{V}_q^{\tau,r}$. We translate the targets in $\mathcal{S}^\tau$ and the projections $P_q^{\tau,r}$, $q \in \mathcal{Q}^{\tau,r}$, into a graph called a view, separately for each r so that information from different relations stays separate during propagation. The membership matrix $\mathbf{M}^{\tau,r} \in \{0, 1\}^{(|\mathcal{S}^\tau| + |\mathcal{Q}^{\tau,r}|) \times |\mathcal{V}^\tau|}$ indicates the members of these subgraphs, with target rows first. A view node representing a target is initialized with the sum of its members' features, and one representing $P_q^{\tau,r}$ with the feature of q. During training, the views contain only training targets; for validation or testing, the corresponding targets are added, and no labels are used as input.

Node-Overlap View When G^τ contains no edges, a subgraph in the subproblem is characterized only by its members. The node-overlap view $\widehat{G}^{\tau,r}_{\text{node}}$ therefore uses the number of shared members, $T_e(S, S') = |\mathcal{V}_S \cap \mathcal{V}_{S'}|$, as the edge weight, giving the weighted adjacency matrix $\widehat{\mathbf{A}}^{\tau,r}_{\text{node}} = \mathbf{M}^{\tau,r}(\mathbf{M}^{\tau,r})^\top$.

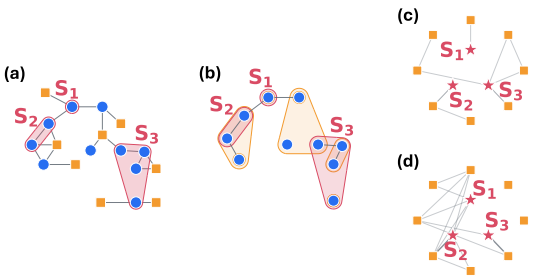

Figure 2: Illustration of constructing a node-overlap view and an edge-connectivity view.

An edge between the nodes representing $P^{\tau,r}_q$ and S^τ_i captures the relation of anchor q to S^τ_i at the set level, and an edge between two targets lets information received by one reach the other. Because targets and anchors exchange information directly through shared members, the view reduces reliance on node-by-node propagation along long paths in G.

Edge-Connectivity View When G^τ contains edges, node membership remains informative, so we keep the node-overlap views and add an edge-connectivity view $\widehat{G}^{\tau,r}_{\text{edge}}$ with the same nodes and initial features. Its edge weight counts pairs of adjacent members, $T_e(S, S') = \sum_{u\in\mathcal{V}_S}\sum_{v\in\mathcal{V}_{S'}} A^\tau_{uv}$, giving $\widehat{\mathbf{A}}^{\tau,r}_{\text{edge}} = \mathbf{M}^{\tau,r}\mathbf{A}^\tau(\mathbf{M}^{\tau,r})^\top$, where $\mathbf{A}^\tau$ is the adjacency matrix of G^τ.

Figure 2 illustrates both views for the example in Figure 1. From (a) to (b), all connected B-type nodes are projected into subgraphs in the A-type subproblem. S2N translation then maps the target and projected subgraphs in (b) to nodes, yielding the node-overlap view in (c) and the edge-connectivity view in (d), which share the same nodes but differ in T_e. Edge width and darkness indicate overlap weight in (c) and connectivity weight in (d).

Same-Type View The node-overlap and edge-connectivity views let anchor nodes receive information from other anchors and target subgraphs, but not from their same-type neighbors in G. When G^τ contains edges, we therefore use it as the same-type view, in which each type-τ node aggregates information from its type-τ neighbors.

5 Method

CTAP uses its views as metapath-based methods use metapath neighborhood graphs: representations are learned on each graph and combined through semantic fusion. Figure 3 illustrates one CTAP layer. Each view is passed to a message-passing neural network (MPNN) layer (Section 5.1); semantic fusion combines the outputs for the circled node (top) or target subgraph (bottom) across views (Section 5.2). Membership pooling incorporates cross-type neighbor features into the node representation and constituent-node features into the target representation (Section 5.3), producing inputs to the next layer. Type-specific target representations remain separate until fusion after the final layer (Section 5.4).

5.1 Message Passing on Views

Let $\mathcal{G}_{\text{view}}$ contain the node-overlap views $\widehat{G}^{\tau,r}_{\text{node}}$ for all τ and $r \in \mathcal{C}^\tau$, together with the edge-connectivity views $\widehat{G}^{\tau,r}_{\text{edge}}$ and the same-type views G^τ for all τ whose G^τ contains edges. Each view $G' \in \mathcal{G}_{\text{view}}$ uses a separately parameterized MPNN at layer $\ell \geq 1$:

$$\widehat{\mathbf{H}}^{(\ell)}_{G'} = \text{MPNN}^{(\ell)}_{G'}(\mathbf{A}_{G'}, \mathbf{H}^{(\ell-1)}_{G'}), \qquad G' \in \mathcal{G}_{\text{view}}, \tag{2}$$

where $\mathbf{A}_{G'}$ is the view adjacency matrix and $\mathbf{H}^{(0)}_{G'} = \mathbf{X}_{G'}$ is its initial feature matrix. For $\ell \geq 1$, the rows of $\mathbf{H}^{(\ell)}_{G'}$ for a target S^τ_i and a node q are their updated representations $\mathbf{h}^{\tau,(\ell)}_{S_i}$ and $\mathbf{h}^{(\ell)}_q$ from Section 5.3, so all views share the current target and node states. We instantiate each MPNN as a GCN layer (Appendix C.7). For translated views, we never form $\widehat{\mathbf{A}}$: we evaluate $\widehat{\mathbf{A}}^{\tau,r}_{\text{node}}\mathbf{H}$ as $\mathbf{M}^{\tau,r}((\mathbf{M}^{\tau,r})^\top\mathbf{H})$ and $\widehat{\mathbf{A}}^{\tau,r}_{\text{edge}}\mathbf{H}$ as $\mathbf{M}^{\tau,r}(\mathbf{A}^\tau((\mathbf{M}^{\tau,r})^\top\mathbf{H}))$, and obtain GCN normalization degrees

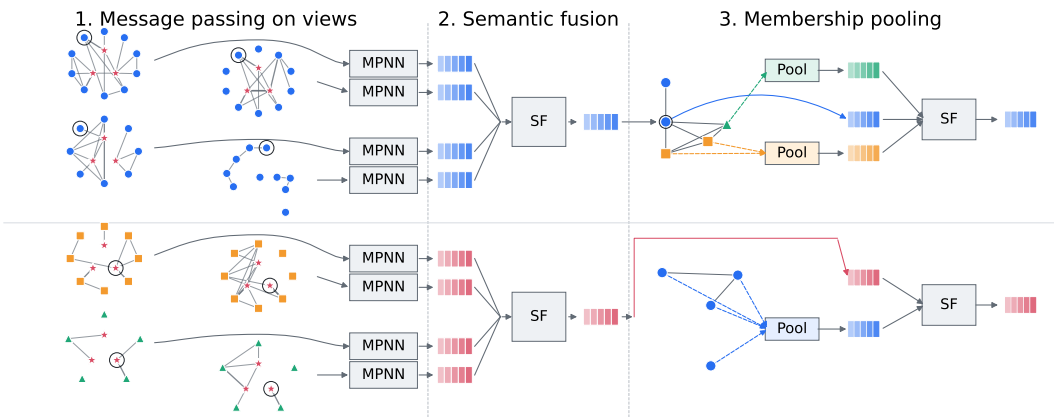

Figure 3: One CTAP layer updating a node (top) and a type-specific target subgraph (bottom), using the example in Figures 1–2. Circles outline the corresponding node in each view: a blue node above and a star-shaped target node below. Dashed arrows indicate the features included in pooling. The rightmost blue and pink bars are vectors representing the node and target subgraph after the update, respectively.

from the same products. The cost thus scales with the numbers of memberships and base edges rather than with the number of translated edges.

5.2 Semantic Fusion of View Outputs

Semantic fusion (SF) combines the rows of an input matrix $\mathbf{H} \in \mathbb{R}^{m \times d}$ into one vector, following the Transformer-based fusion of SeHGNN (Yang et al., 2023):

$$\mathrm{SF}(\mathbf{H}) = \mathbf{W}_O \,\mathrm{vec}\left(\mathbf{H} + \gamma \mathbf{P}\mathbf{H}\mathbf{W}_V\right), \qquad \mathbf{P} = \mathrm{softmax}\left(\mathbf{H}\mathbf{W}_Q(\mathbf{H}\mathbf{W}_K)^\top / \sqrt{d_k}\right), \tag{3}$$

where $\mathrm{vec}(\cdot)$ concatenates rows, $\mathbf{W}_Q, \mathbf{W}_K \in \mathbb{R}^{d \times d_k}$, $\mathbf{W}_V \in \mathbb{R}^{d \times d}$, and $\mathbf{W}_O$ are learnable, and the learnable scalar γ is initialized to zero. Each row occupies a fixed position assigned to its source, such as a view, and rows of unavailable sources are zero-padded and masked.

For each node $q \in \mathcal{V}$, the rows of $\mathbf{H}_q^{(\ell)}$ are the MPNN outputs for q from every view containing it: the node-overlap and edge-connectivity views in which q is an anchor, and the same-type view of its type. For each target subgraph S_i and type τ with $S_i^\tau \in \mathcal{S}^\tau$, the rows of $\mathbf{H}_{S_i}^{\tau,(\ell)}$ are the outputs for S_i^τ from the node-overlap and edge-connectivity views of the τ-type subproblem. We fuse these input matrices as

$$\mathbf{u}_q^{(\ell)} = \mathrm{SF}^{(\ell)}_{\mathrm{view,node}}(\mathbf{H}_q^{(\ell)}), \qquad \mathbf{u}_{S_i}^{\tau,(\ell)} = \mathrm{SF}^{(\ell)}_{\mathrm{view,target}}(\mathbf{H}_{S_i}^{\tau,(\ell)}), \tag{4}$$

where the two SF modules are separately parameterized at each layer.

5.3 Membership Pooling

Membership pooling incorporates features of neighboring nodes of other types into each node representation and features of constituent nodes into each subgraph representation. We pool these node features separately for each membership set and combine the pooled features with the view-fused representation using semantic fusion.

For node q, let $\mathcal{R}_q$ be the set of cross-type relations incident to q, each $r \in \mathcal{R}_q$ connecting q to nodes of some type τ. We pool the representations of its neighbors in $\mathcal{V}_q^{\tau,r}$ for each $r \in \mathcal{R}_q$:

$$\mathbf{c}_q^{\tau,r,(\ell)} = \mathrm{Pool}(\{\mathbf{u}_v^{(\ell)} \mid v \in \mathcal{V}_q^{\tau,r}\}), \tag{5}$$

where $\mathrm{Pool}(\cdot)$ is mean, max, or size pooling (the sum divided by the square root of the set size), selected by hyperparameter search. Stacking these vectors as the columns of $\mathbf{C}_q^{(\ell)}$ in a fixed relation order, we fuse them with the view-fused representation of q:

$$\mathbf{h}_q^{(\ell)} = \mathrm{SF}^{(\ell)}_{\mathrm{member,node}}([\mathbf{u}_q^{(\ell)}, \mathbf{C}_q^{(\ell)}]^\top). \tag{6}$$

Table 1: Test micro-F1 over ten runs, reported as mean ± standard deviation. Bold and underline indicate the highest and second-highest means, respectively. Avg. rank averages ranks over six datasets, averaging ties; lower is better. † indicates a higher CTAP mean than the best baseline at $p < 0.01$ under Welch's unpaired t-test described in Appendix G.6.

Model	DBLP	Movie	Rare Disease Category	Rare Disease Inheritance	Protein Set Function	Gene Set Process	Avg. rank ↓
SeHGNNseg + HINPool	0.171 ± 0.029	0.121 ± 0.023	0.924 ± 0.012	0.889 ± 0.014	0.474 ± 0.098	0.493 ± 0.008	7.33
EIGNNplain	0.286 ± 0.007	0.387 ± 0.022	$\underline{0.972 \pm 0.005}$	0.900 ± 0.019	0.740 ± 0.017	0.592 ± 0.013	3.67
EIGNNplain (metapath)	0.267 ± 0.052	0.381 ± 0.093	$\mathbf{0.974 \pm 0.003}$	$\underline{0.912 \pm 0.003}$	0.745 ± 0.016	0.595 ± 0.011	3.50
GLASS + SeHGNN	0.131 ± 0.009	0.219 ± 0.058	0.707 ± 0.139	0.828 ± 0.020	0.419 ± 0.027	0.510 ± 0.008	8.33
WLKS-$\{0, D\}$	$\underline{0.293 \pm 0.000}$	0.432 ± 0.000	0.930 ± 0.000	0.861 ± 0.000	0.762 ± 0.000	$\underline{0.601 \pm 0.000}$	3.83
GPEN	0.243 ± 0.014	0.443 ± 0.017	0.757 ± 0.039	0.736 ± 0.027	0.626 ± 0.045	0.555 ± 0.014	6.50
SubGNN	0.285 ± 0.009	$\underline{0.477 \pm 0.005}$	0.932 ± 0.010	0.865 ± 0.008	$\underline{0.813 \pm 0.041}$	0.548 ± 0.094	4.00
Rad et al.	0.262 ± 0.009	0.366 ± 0.028	0.950 ± 0.009	0.833 ± 0.045	0.712 ± 0.036	0.473 ± 0.034	6.50
CTAP (ours)	$\mathbf{0.300 \pm 0.007}$	$\mathbf{0.586 \pm 0.009}^{\dagger}$	0.967 ± 0.007	$\mathbf{0.935 \pm 0.002}^{\dagger}$	$\mathbf{0.821 \pm 0.027}$	$\mathbf{0.618 \pm 0.011}^{\dagger}$	1.33

For target subgraph S_i and each node type τ with $S_i^\tau \in \mathcal{S}^\tau$, we pool the updated representations of its constituent nodes and fuse the result with the view-fused target representation:

$$\mathbf{c}_{S_i}^{\tau,(\ell)} = \mathrm{Pool}(\{\mathbf{h}_v^{(\ell)} \mid v \in \mathcal{V}_{S_i}^\tau\}), \qquad \mathbf{h}_{S_i}^{\tau,(\ell)} = \mathrm{SF}_{\mathrm{member,target}}^{(\ell)}([\mathbf{u}_{S_i}^{\tau,(\ell)}, \mathbf{c}_{S_i}^{\tau,(\ell)}]^\top). \tag{7}$$

Both SF modules are separately parameterized at each layer. With fixed numbers of node types and relations, one CTAP layer costs $O(d(|\mathcal{E}| + \sum_i |\mathcal{V}_{S_i}|) + d^2(|\mathcal{V}| + |\mathcal{S}|))$, linear in the base graph size and the total size of the targets (Appendix D.7).

5.4 Subgraph Representation

After L layers, we fuse the final target representations across node types to represent S_i:

$$\mathbf{h}_{S_i} = \mathrm{SF}_{\mathrm{subgraph}}(\mathbf{H}_{S_i}^{(L)}), \tag{8}$$

where the rows of $\mathbf{H}_{S_i}^{(L)}$ are $\mathbf{h}_{S_i}^{\tau,(L)}$ in a fixed order of node types, with rows of types absent from S_i zero-padded and masked, and $\mathrm{SF}_{\mathrm{subgraph}}$ is a separately parameterized SF module. For subgraph classification, we compute logits as $\hat{\mathbf{y}}_{S_i} = f_{\mathrm{cls}}(\mathbf{h}_{S_i})$ and jointly train CTAP and the classification head end-to-end by minimizing the softmax cross-entropy for single-label tasks or the class-averaged sigmoid binary cross-entropy for multi-label tasks, averaged over the training targets. Our implementation also considers Jumping Knowledge concatenation across layers, as detailed in Appendix C.13. Minibatches determine only which training targets contribute to the loss; message passing always runs on views containing all training targets.

6 Experiments

All methods use the same pretrained node features and data splits, are tuned per dataset with 50 Optuna trials by validation micro-F1, and are evaluated over ten runs with different seeds (Appendix I).

6.1 Datasets and Baseline Methods

Datasets DBLP and Movie, built from data used in team discovery research, predict a paper's venue from its author or term set and a movie's genres (multi-label) from its actor or keyword set. Rare Disease Category (RD-C) and Rare Disease Inheritance (RD-I), built from the patient records released with SHEPHERD (Alsentzer et al., 2025), predict a patient's disease category and inheritance patterns (multi-label), respectively, from a subgraph of the patient's observed phenotypes and causal gene in a biomedical knowledge graph. Protein Set Function (PSF) and Gene Set Process (GSP) predict the functional class of a Reactome protein set and the biological-process category of a gene/protein set annotated with Gene Ontology (GO) terms. Venue nodes, genre edges, and label pathway or GO-term nodes are removed from the input graphs to prevent label leakage. Appendix F details dataset construction and statistics.

Table 2: Test micro-F1 of CTAP ablations over ten repeated runs. Values are reported as mean ± standard deviation. Bold indicates the highest value in each dataset. N/A indicates that the view ablation is not applicable.

Model	DBLP	Movie	Rare Disease Category	Rare Disease Inheritance	Protein Set Function	Gene Set Process
CTAP (full)	$\mathbf{0.300 \pm 0.007}$	$\mathbf{0.586 \pm 0.009}$	0.967 ± 0.007	$\mathbf{0.935 \pm 0.002}$	$\mathbf{0.821 \pm 0.027}$	$\mathbf{0.618 \pm 0.011}$
w/o node-overlap view	N/A	N/A	0.955 ± 0.007	0.914 ± 0.011	0.787 ± 0.036	0.526 ± 0.010
w/o edge-connectivity view	N/A	N/A	0.965 ± 0.009	0.932 ± 0.005	0.814 ± 0.023	0.594 ± 0.010
w/o same-type view	N/A	N/A	0.951 ± 0.012	0.924 ± 0.005	0.758 ± 0.023	0.595 ± 0.008
w/o membership pooling	0.281 ± 0.008	0.545 ± 0.009	0.965 ± 0.006	0.927 ± 0.006	0.811 ± 0.014	0.605 ± 0.009
w/o cross-type projection	0.229 ± 0.007	0.313 ± 0.028	0.937 ± 0.008	0.883 ± 0.021	0.514 ± 0.062	0.559 ± 0.012
w/o type decomposition	0.165 ± 0.005	0.246 ± 0.049	0.572 ± 0.021	0.310 ± 0.040	0.426 ± 0.016	0.546 ± 0.011
SF $\rightarrow$ mean	0.291 ± 0.006	0.581 ± 0.004	0.951 ± 0.006	0.918 ± 0.002	0.807 ± 0.022	0.593 ± 0.015
SF $\rightarrow$ concat	0.299 ± 0.006	0.576 ± 0.007	$\mathbf{0.969 \pm 0.005}$	0.926 ± 0.003	0.791 ± 0.030	0.603 ± 0.013

Baseline Methods GNNseg and GNNplain encode nodes within each target subgraph and on the shared base graph, respectively, then pool the target's node embeddings; we use GCN (Kipf & Welling, 2017), GATv2 (Brody et al., 2022), HGNN (Lv et al., 2021; Yang et al., 2023), and EIGNN (Liu et al., 2021) backbones to examine heterogeneity handling and long-range dependencies. Subgraph baselines include SubGNN (Alsentzer et al., 2020), its heterogeneous extension by Hamidi Rad et al. (2023), GLASS (Wang & Zhang, 2022), SSNP (Jacob et al., 2023), ISNN (Zhong et al., 2025), S2N (Kim & Oh, 2024), GPEN (Wu et al., 2025), and WLKS-$\{0, D\}$ (Kim & Oh, 2025). We also instantiate GLASS with HGNN backbones and extend HGNN-based baselines with HINPool (Hong et al., 2026) to test heterogeneity handling with subgraph membership labels and type-specific pooling, respectively. Appendix H details the baseline implementations.

6.2 Results

Main Results Table 1 reports the best-performing variant of each baseline family, and Appendix G reports all baseline results. CTAP achieves the highest mean performance on five of the six datasets. Compared with the highest-mean baselines in Table 1, its gains are approximately 0.7, 10.9, 2.3, and 1.7 percentage points (pp) on DBLP, Movie, Rare Disease Inheritance, and Gene Set Process, respectively. On Protein Set Function, CTAP performs comparably to SubGNN.

On Rare Disease Category, CTAP trails EIGNNplain (metapath) by 0.7 pp, comparable to CTAP's run-to-run standard deviation and not significant at the 1% level under Welch's unpaired t-test in Appendix G.6. EIGNNplain also reaches 0.972, indicating that base-graph propagation capturing long-range dependencies already provides a strong predictive signal on this task and leaves limited room for improvement.

Ablation Study Table 2 evaluates CTAP's components and alternatives to semantic fusion; Appendix K.1 details each variant. All ablations except concatenation on Category yield lower mean performance than full CTAP. Removing node-overlap views causes the largest view-ablation drop on Gene Set Process (9.2 pp), highlighting shared membership over node connectivity. Removing edge-connectivity views lowers Gene Set Process by 2.4 pp, indicating functional information beyond shared membership. Protein Set Function drops most without same-type views (6.3 pp), highlighting protein interactions for collective function. The 2.3 pp drop on Gene Set Process indicates the role of same-type relations outside target constituents. Without membership pooling, Movie and DBLP lose 4.1 and 1.9 pp, highlighting pooling where alternative views are unavailable. Smaller drops elsewhere suggest compensation by other views. Without cross-type projection, Movie and Protein Set Function lose 27.3 and 30.7 pp: target overlap or protein interactions alone cannot sustain full performance. Removing type decomposition lowers Category and Inheritance by 39.5 and 62.5 pp, highlighting distinct gene/phenotype roles and type-specific relations. Gene Set Process loses less, suggesting that shared membership may partly compensate. Mean aggregation loses 0.5–2.5 pp against SF on all six datasets; concatenation loses 0.1–3.0 pp on five but gains 0.2 pp on Category. SF's modest gains require 1.15–$1.30\times$ the alternatives' inference time on the two measured datasets under fixed configurations. Appendix G.7 details this comparison.

Performance with Reduced Training Data Figure 4 compares performance using 10–50% and 100% of the training data. CTAP outperforms EIGNNplain (metapath) at every reduced-data level on DBLP, Movie, and Rare Disease Inheritance. On Inheritance, the gap grows from 2.3 pp with all training data to 17.2 pp with 10%, suggesting greater benefits under limited supervision. On

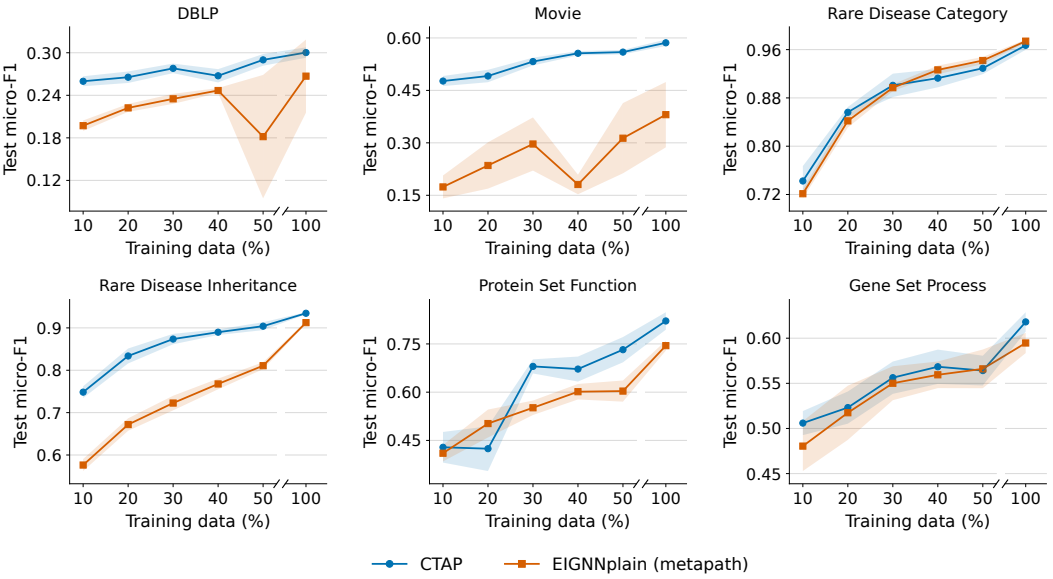

Figure 4: Test micro-F1 with varying training-data proportions. Curves show means; shaded regions indicate standard deviations.

Table 3: Computational cost on Movie and Rare Disease Category. Times are in seconds; GPU memory is in MiB. SubGNN preprocessing values are per-configuration estimates.

Dataset	Model	Train/epoch	Inference	Memory	Preprocessing
Movie	CTAP	10.487	0.214	902	11.730
	SubGNN	358.286	30.000	1,460	1823.170
	GPEN	9.527	0.089	928	1.560
	EIGNNplain (metapath)	4.314	0.001	607	0.480
	SeHGNNseg + HINPool	17.835	0.961	862	4.740
RD Category	CTAP	46.541	0.551	1,481	59.650
	SubGNN	19.562	1.400	1,202	770.330
	GPEN	17.644	0.048	2,601	2.730
	EIGNNplain (metapath)	4.576	0.001	934	0.310
	SeHGNNseg + HINPool	23.702	1.360	546	9.260

Movie, CTAP trained with 10% outperforms the baseline trained with 100%. CTAP leads at 10–30% on Category and at all levels except 20% on Protein Set Function; Gene Set Process shows small differences.

6.3 Efficiency

Table 3 compares computational costs on Movie and Rare Disease Category; measurement details are provided in Appendix K.2. On Movie, CTAP's training time is close to GPEN's and lower than SubGNN's and SeHGNNseg + HINPool's; on Category, it is the highest. On both datasets, CTAP predicts the complete test split in under a second, faster than SubGNN and SeHGNNseg + HINPool, with lower memory use than GPEN. Preprocessing is slower than for all baselines except SubGNN.

CTAP performs propagation through membership products without constructing translated adjacencies, processing approximately 19–163$\times$ fewer nonzero entries per layer than explicit adjacency-based propagation. This reduces the sparse propagation cost despite the 48.60–387.32$\times$ increase in view edges over the base graph, as quantified in Appendix E. GCN normalization is also computed without constructing translated adjacencies, as detailed in Appendix C.7.

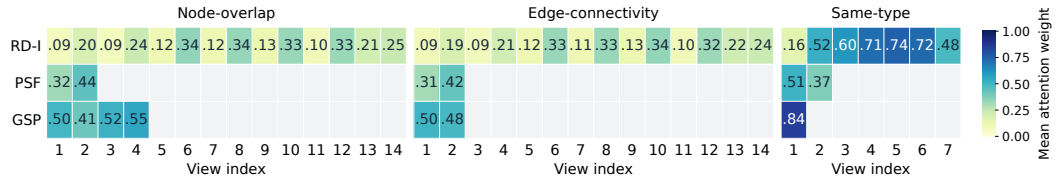

Figure 5: Mean view attention before membership pooling. RD-I, PSF, and GSP denote the Rare Disease Inheritance, Protein Set Function, and Gene Set Process datasets, respectively. View indices are dataset-specific; gray cells indicate absent views.

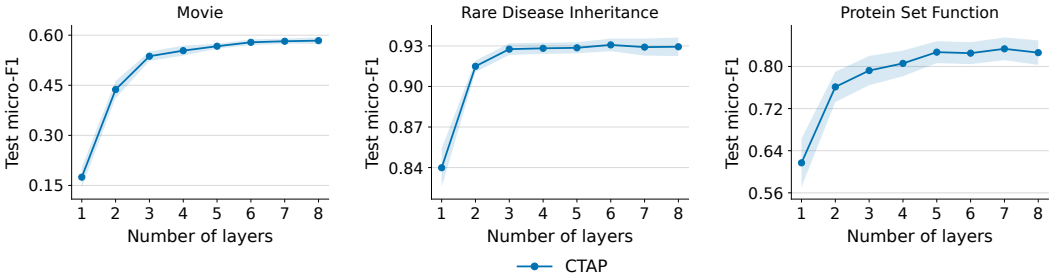

Figure 6: Test micro-F1 with varying numbers of CTAP layers. Curves show means over ten runs; shaded regions indicate standard deviations.

6.4 Analysis

Figure 5 shows view attention on Inheritance, Protein, and Gene. Category shows similar patterns to Inheritance and is omitted here. Appendix K.3 provides their comparison, DBLP/Movie results, and the aggregation procedure. On both disease tasks, disease, process, function, and component same-type views receive approximately 0.60–0.74; same-type views also account for approximately 0.54 when averaged over all nodes. This weight suggests a strong predictive signal from the base-graph relations discussed in Section 6.2. At gene/protein nodes, translated views receive approximately 0.86 versus 0.14 for same-type views, indicating substantial use of projected-subgraph relations in gene representations. Phenotype nodes instead assign approximately 0.52–0.53 to same-type views, suggesting that the relations emphasized differ by node type. On Protein Set Function, attention at protein nodes is split between protein–protein relations (0.510) and translated views (0.490). On Gene Set Process, node-overlap and edge-connectivity attention are comparable at gene nodes (0.496 versus 0.504) and protein nodes (0.515 versus 0.485).

Figure 6 shows CTAP's performance across layer counts. All three datasets improve substantially from one to three layers. Movie continues to improve through eight layers, whereas Inheritance largely plateaus after three. Protein improves overall through seven layers; the decrease at eight is smaller than its run-to-run standard deviation. Appendix K.4 details the experimental setup.

7 Conclusion

We presented CTAP for subgraph representation learning on heterogeneous graphs. It separates the target subgraphs and their shared base graph by node type and projects connected nodes of other types into each resulting subproblem. The node-overlap, edge-connectivity, and same-type views capture complementary relationships, while semantic fusion and membership pooling combine their outputs with information from cross-type neighbors and constituent nodes. Each target subgraph's representations are then combined across node types. CTAP achieves the highest mean test micro-F1 on five of six datasets among the compared methods, with ablations supporting its main components. Propagation through membership products reduces sparse computation without explicitly constructing translated adjacencies. These results support organizing subgraph representation learning around both subgraph membership and the types and relations of the shared base graph. Appendix L discusses limitations and future directions.

Reproducibility Statement

To support reproducibility, we provide the source code for CTAP, baseline comparisons, and additional experiments, together with data preprocessing scripts, experiment configurations, and execution instructions, at https://anonymous.4open.science/r/CTAP-Cross-Type-Anchor-Projection--BB41/. The CTAP architecture is described in Section 5, with implementation details in Appendix C. Section 6.1 introduces the datasets and baseline methods, and Appendix F details dataset construction. Training and evaluation procedures and hyperparameter selection are documented in Appendices I and J, respectively.

AI Use Statement

The authors developed the research ideas and method design. Generative AI tools assisted with manuscript drafting and review, figure preparation, code implementation and debugging, and discussion of result interpretations. The authors take responsibility for the final manuscript and accompanying code.

Ethics Statement

The authors have read and adhered to the ICLR Code of Ethics. This research develops a method for subgraph representation learning using datasets constructed from existing sources, as detailed in Appendix F. No new human participants were recruited, and no clinical interventions were conducted. The rare disease experiments evaluate subgraph classification rather than clinical deployment. The goal of this research is to advance representation learning on heterogeneous graphs.

References

Bijaya Adhikari, Yao Zhang, Naren Ramakrishnan, et al. Sub2Vec: Feature learning for subgraphs. In *Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, pp. 170–182. Springer International Publishing, 2018. doi: 10.1007/978-3-319-93037-4_14. URL https://doi.org/10.1007/978-3-319-93037-4_14.

Takuya Akiba, Shotaro Sano, Toshihiko Yanase, et al. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pp. 2623–2631. ACM, 2019. doi: 10.1145/3292500.3330701. URL https://doi.org/10.1145/3292500.3330701.

Suzi A Aleksander, James Balhoff, Seth Carbon, J Michael Cherry, Harold J Drabkin, Dustin Ebert, Marc Feuermann, Pascale Gaudet, Nomi L Harris, et al. The Gene Ontology knowledgebase in 2023. *Genetics*, 224(1):iyad031, 2023. doi: 10.1093/genetics/iyad031. URL https://doi.org/10.1093/genetics/iyad031.

Emily Alsentzer, Samuel Finlayson, Michelle Li, et al. Subgraph neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pp. 8017–8029. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/5bca8566db79f3788be9efd96c9ed70d-Paper.pdf.

Emily Alsentzer, Michelle M. Li, Shilpa N. Kobren, et al. Few shot learning for phenotype-driven diagnosis of patients with rare genetic diseases. *npj Digital Medicine*, 8(1):380, 2025. doi: 10.1038/s41746-025-01749-1. URL https://doi.org/10.1038/s41746-025-01749-1.

Rounak Banik. The movies dataset. Kaggle, 2017. URL https://www.kaggle.com/datasets/rounakbanik/the-movies-dataset. Accessed: 2026-06-29.

D. Barrell, E. Dimmer, R. P. Huntley, et al. The GOA database in 2009–an integrated Gene Ontology Annotation resource. *Nucleic Acids Research*, 37(Database):D396–D403, 2009. doi: 10.1093/nar/gkn803. URL https://doi.org/10.1093/nar/gkn803.

Shaked Brody, Uri Alon, and Eran Yahav. How attentive are graph attention networks? In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2022. URL https://openreview.net/forum?id=F72ximsx7C1.

Tianle Cai, Shengjie Luo, Keyulu Xu, et al. GraphNorm: A principled approach to accelerating graph neural network training. In *Proceedings of the International Conference on Machine Learning (ICML)*, volume 139 of *Proceedings of Machine Learning Research*, pp. 1204–1215. PMLR, 2021. URL https://proceedings.mlr.press/v139/cai21e.html.

Jeffrey T. Chang. Deriving transcriptional programs and functional processes from gene expression databases. *Bioinformatics*, 28(8):1122–1129, 2012. doi: 10.1093/bioinformatics/bts112. URL https://doi.org/10.1093/bioinformatics/bts112.

Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019. URL https://arxiv.org/abs/1903.02428.

Xinyu Fu, Jiani Zhang, Ziqiao Meng, et al. MAGNN: Metapath aggregated graph neural network for heterogeneous graph embedding. In *Proceedings of The Web Conference (WWW)*, pp. 2331–2341. ACM, 2020. doi: 10.1145/3366423.3380297. URL https://doi.org/10.1145/3366423.3380297.

Fangda Gu, Heng Chang, Wenwu Zhu, et al. Implicit graph neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pp. 11984–11995. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/8b5c8441a8ff8e151b191c53c1842a38-Paper.pdf.

Radin Hamidi Rad, Hoang Nguyen, Feras Al-Obeidat, et al. Learning heterogeneous subgraph representations for team discovery. *Information Retrieval Journal*, 26(1–2):8, 2023. doi: 10.1007/s10791-023-09421-6. URL https://doi.org/10.1007/s10791-023-09421-6.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/5dd9db5e033da9c6fb5ba83c7a7ebea9-Paper.pdf.

Ming-Yi Hong, You-Chen Teng, Shao-En Lin, et al. HINPool: A unified heterogeneous graph pooling framework for accurate molecular and protein property prediction. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, volume 40, pp. 21752–21760. Association for the Advancement of Artificial Intelligence (AAAI), 2026. doi: 10.1609/aaai.v40i26.39326. URL https://doi.org/10.1609/aaai.v40i26.39326.

Ziniu Hu, Yuxiao Dong, Kuansan Wang, et al. Heterogeneous graph transformer. In *Proceedings of The Web Conference (WWW)*, pp. 2704–2710. ACM, 2020. doi: 10.1145/3366423.3380027. URL https://doi.org/10.1145/3366423.3380027.

Shweta Ann Jacob, Paul Louis, and Amirali Salehi-Abari. Stochastic subgraph neighborhood pooling for subgraph classification. In *Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM)*, pp. 3963–3967. ACM, 2023. doi: 10.1145/3583780.3615227. URL https://doi.org/10.1145/3583780.3615227.

Ying Ji, Guojia Wan, Yibing Zhan, et al. Metapath-fused heterogeneous graph network for molecular property prediction. *Information Sciences*, 629:155–168, 2023. doi: 10.1016/j.ins.2023.01.092. URL https://doi.org/10.1016/j.ins.2023.01.092.

Dongkwan Kim and Alice Oh. Translating subgraphs to nodes makes simple GNNs strong and efficient for subgraph representation learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, volume 235 of *Proceedings of Machine Learning Research*, pp. 24415–24436. PMLR, 2024. URL https://proceedings.mlr.press/v235/kim24aa.html.

Dongkwan Kim and Alice Oh. Generalizing Weisfeiler-Lehman kernels to subgraphs. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2025. URL https://openreview.net/forum?id=HZgZrtIreg.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015. URL https://arxiv.org/abs/1412.6980.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017. URL https://arxiv.org/abs/1609.02907.

Sebastian Köhler, Michael Gargano, Nicolas Matentzoglu, et al. The Human Phenotype Ontology in 2021. *Nucleic Acids Research*, 49(D1):D1207–D1217, 2021. doi: 10.1093/nar/gkaa1043. URL https://doi.org/10.1093/nar/gkaa1043.

David M Levine, David R Haynor, John C Castle, et al. Pathway and gene-set activation measurement from mRNA expression data: the tissue distribution of human pathways. *Genome Biology*, 7(10):R93, 2006. doi: 10.1186/gb-2006-7-10-r93. URL https://doi.org/10.1186/gb-2006-7-10-r93.

Michael Ley. The DBLP computer science bibliography: Evolution, research issues, perspectives. In *Proceedings of the International Symposium on String Processing and Information Retrieval (SPIRE)*, pp. 1–10. Springer Berlin Heidelberg, 2002. doi: 10.1007/3-540-45735-6_1. URL https://doi.org/10.1007/3-540-45735-6_1.

Chang Liu, Yuwen Yang, Zhe Xie, et al. Position-aware subgraph neural networks with data-efficient learning. In *Proceedings of the ACM International Conference on Web Search and Data Mining (WSDM)*, pp. 643–651. ACM, 2023. doi: 10.1145/3539597.3570429. URL https://doi.org/10.1145/3539597.3570429.

Juncheng Liu, Kenji Kawaguchi, Bryan Hooi, et al. EIGNN: Efficient infinite-depth graph neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 34, pp. 18762–18773. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/9bd5ee6fe55aaeb673025dbcb8f939c1-Paper.pdf.

Qingsong Lv, Ming Ding, Qiang Liu, et al. Are we really making much progress? Revisiting, benchmarking and refining heterogeneous graph neural networks. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pp. 1150–1160. ACM, 2021. doi: 10.1145/3447548.3467350. URL https://doi.org/10.1145/3447548.3467350.

Changping Meng, S Chandra Mouli, Bruno Ribeiro, et al. Subgraph pattern neural networks for high-order graph evolution prediction. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, volume 32. Association for the Advancement of Artificial Intelligence (AAAI), 2018. doi: 10.1609/aaai.v32i1.11747. URL https://doi.org/10.1609/aaai.v32i1.11747.

Marija Milacic, Deidre Beavers, Patrick Conley, et al. The Reactome pathway knowledgebase 2024. *Nucleic Acids Research*, 52(D1):D672–D678, 2024. doi: 10.1093/nar/gkad1025. URL https://doi.org/10.1093/nar/gkad1025.

MyGene2. Website aims to accelerate gene discovery, diagnosis, treatment: MyGene2.org fosters open sharing among families, researchers, and clinicians. *American Journal of Medical Genetics Part A*, 170(6):1388–1389, 2016. doi: 10.1002/ajmg.a.37746. URL https://doi.org/10.1002/ajmg.a.37746.

Ana Rath, Annie Olry, Ferdinand Dhombres, et al. Representation of rare diseases in health information systems: The Orphanet approach to serve a wide range of end users. *Human Mutation*, 33(5):803–808, 2012. doi: 10.1002/humu.22078. URL https://doi.org/10.1002/humu.22078.

Michael Schlichtkrull, Thomas N. Kipf, Peter Bloem, et al. Modeling relational data with graph convolutional networks. In *Proceedings of the European Semantic Web Conference (ESWC)*, pp. 593–607. Springer International Publishing, 2018. doi: 10.1007/978-3-319-93417-4_38. URL https://doi.org/10.1007/978-3-319-93417-4_38.

Varun S Sharma, Andrea Fossati, Rodolfo Ciuffa, et al. PCfun: a hybrid computational framework for systematic characterization of protein complex function. *Briefings in Bioinformatics*, 23(4): bbac239, 2022. doi: 10.1093/bib/bbac239. URL https://doi.org/10.1093/bib/bbac239.

The Movie Database. The Movie Database. Online database, 2026. URL https://www.themoviedb.org/. Accessed: 2026-06-29.

Xiao Wang, Houye Ji, Chuan Shi, et al. Heterogeneous graph attention network. In *Proceedings of The Web Conference (WWW)*, pp. 2022–2032. ACM, 2019. doi: 10.1145/3308558.3313562. URL https://doi.org/10.1145/3308558.3313562.

Xiyuan Wang and Muhan Zhang. GLASS: GNN with labeling tricks for subgraph representation learning. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2022. URL https://openreview.net/forum?id=XLxhEjKNbXj.

B. L. Welch. The generalization of 'Student's' problem when several different population variances are involved. *Biometrika*, 34(1–2):28–35, 1947. doi: 10.1093/biomet/34.1-2.28. URL https://doi.org/10.1093/biomet/34.1-2.28.

Chuhan Wu, Fangzhao Wu, Yongfeng Huang, et al. User-as-Graph: User modeling with heterogeneous graph pooling for news recommendation. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1624–1630. International Joint Conferences on Artificial Intelligence Organization, 2021. doi: 10.24963/ijcai.2021/224. URL https://doi.org/10.24963/ijcai.2021/224.

Nannan Wu, Yuming Huang, Yiming Zhao, et al. GPEN: Global position encoding network for enhanced subgraph representation learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, volume 267 of *Proceedings of Machine Learning Research*, pp. 67326–67341. PMLR, 2025. URL https://proceedings.mlr.press/v267/wu25l.html.

Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 5453–5462. PMLR, 2018. URL https://proceedings.mlr.press/v80/xu18c.html.

Bishan Yang, Scott Wen-tau Yih, Xiaodong He, et al. Embedding entities and relations for learning and inference in knowledge bases. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015. URL https://www.microsoft.com/en-us/research/publication/embedding-entities-and-relations-for-learning-and-inference-in-knowledge-bases/.

Xiaocheng Yang, Mingyu Yan, Shirui Pan, et al. Simple and efficient heterogeneous graph neural network. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, volume 37, pp. 10816–10824. Association for the Advancement of Artificial Intelligence (AAAI), 2023. doi: 10.1609/aaai.v37i9.26283. URL https://doi.org/10.1609/aaai.v37i9.26283.

Kexin Zhang, Feng Huang, Luotao Liu, et al. Heterogeneous causal metapath graph neural network for gene-microbe-disease association prediction. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 6161–6169. International Joint Conferences on Artificial Intelligence Organization, 2024. doi: 10.24963/ijcai.2024/681. URL https://doi.org/10.24963/ijcai.2024/681.

Yongjian Zhong, Liao Zhu, Hieu Vu, et al. Implicit subgraph neural network. In *Proceedings of the International Conference on Machine Learning (ICML)*, volume 267 of *Proceedings of Machine Learning Research*, pp. 78602–78615. PMLR, 2025. URL https://proceedings.mlr.press/v267/zhong25g.html.

Qiuyu Zhu, Liang Zhang, Qianxiong Xu, et al. HHGT: Hierarchical heterogeneous graph transformer for heterogeneous graph representation learning. In *Proceedings of the ACM International Conference on Web Search and Data Mining (WSDM)*, pp. 318–326. ACM, 2025. doi: 10.1145/3701551.3703511. URL https://doi.org/10.1145/3701551.3703511.

A NOTATION

Table 4 summarizes the notation used in the main text and appendices. Each description follows the definition of the symbol in the main text or appendices.

Table 4: Notation used in this paper.

Symbol	Description
Graphs and problem setting	
$G = (\mathcal{V}, \mathcal{E}, \mathbf{X})$	G is the undirected base graph. $\mathcal{V}$ is its finite node set, $\mathcal{E}$ is its edge set, and $\mathbf{X}$ is its node feature matrix.
u, v	Nodes in G.
q	A node of type κ connected to nodes of type τ in cross-type projection.
v_i	The i-th node of $\mathcal{V}$.
$\mathbf{A}$	Adjacency matrix of G.
$\mathbf{A}_{uv}$	Weight of edge $\{v_u, v_v\}$ if it exists, and zero otherwise.
$\mathbf{X}_i$	$\mathbf{X}_i$ is the i-th row of $\mathbf{X}$ and represents the feature vector of the i-th node v_i.
$\mathcal{T}_\mathcal{V}$	Set of node types in the heterogeneous graph.
$\mathcal{T}_\mathcal{E}$	Set of edge types in the heterogeneous graph.
$f_\mathcal{V}$	Function assigning a node type to each node in $\mathcal{V}$.
$f_\mathcal{E}$	Function assigning an edge type to each edge in $\mathcal{E}$.
τ	$\tau \in \mathcal{T}_\mathcal{V}$ is the node type for which a subproblem is formed.
κ	$\kappa \neq \tau$ is the type of node q whose projection is used in the subproblem for node type τ.
r	$r \in \mathcal{T}_\mathcal{E}$ is an edge type connecting nodes of types τ and κ.
$S = (\mathcal{V}_S, \mathcal{E}_S, \mathbf{X}_S)$	S is a subgraph of G. $\mathcal{V}_S$ contains its nodes, $\mathcal{E}_S$ contains its edges, and $\mathbf{X}_S$ contains the rows of $\mathbf{X}$ corresponding to $\mathcal{V}_S$ under the ordering inherited from G.
$\mathcal{S} = \{S_1, \ldots, S_n\}$	Collection of subgraphs (targets) given with the base graph G.
n	Number of subgraphs in $\mathcal{S}$.
d	Input feature dimension in the problem setting; feature dimension in message passing.
d'	Dimension of the learned subgraph representation.
d_0	Initial feature width in the complexity analysis of view construction.
$\mathcal{S}_{\text{train}}$, $\mathcal{S}_{\text{val}}$, $\mathcal{S}_{\text{test}}$	Sets of training, validation, and test subgraphs, respectively.
$\mathcal{S}_B$	$\mathcal{S}_B \subseteq \mathcal{S}_{\text{train}}$ is a training minibatch.
Type-aware decomposition and cross-type projection	
$G^\tau = (\mathcal{V}^\tau, \mathcal{E}^\tau, \mathbf{X}^\tau)$	G^τ is obtained from G by retaining nodes of type τ and the edges among them. $\mathcal{V}^\tau$ and $\mathcal{E}^\tau$ are these node and edge sets, and $\mathbf{X}^\tau$ contains their node features.
$S_i^\tau = (\mathcal{V}_{S_i}^\tau, \mathcal{E}_{S_i}^\tau, \mathbf{X}_{S_i}^\tau)$	S_i^τ is the part of subgraph S_i obtained by retaining its nodes of type τ and the edges among them. $\mathcal{V}_{S_i}^\tau$ and $\mathcal{E}_{S_i}^\tau$ are these node and edge sets, and $\mathbf{X}_{S_i}^\tau$ contains their node features.
$\mathcal{S}^\tau$	Set of nonempty subgraphs S_i^τ obtained from subgraphs $S_i \in \mathcal{S}$ by retaining their nodes of type τ and the edges among them.
$\mathcal{S}_{\text{train}}^\tau$, $\mathcal{S}_{\text{val}}^\tau$, $\mathcal{S}_{\text{test}}^\tau$	Sets of nonempty subgraphs obtained from training, validation, and test subgraphs, respectively, by retaining their nodes of type τ and the edges among them.
$\mathcal{C}^\tau$	Set of cross-type relations incident to node type τ.
$\mathcal{R}_q$	Set of cross-type relations incident to node q.
$\mathcal{V}_q^{\tau,r}$	$\mathcal{V}_q^{\tau,r}$ is the set of nodes of type τ adjacent to q through edges of type r.
$P_q^{\tau,r}$	$P_q^{\tau,r}$ is the projection of node q through edge type r into the subproblem for node type τ, defined as the subgraph induced by $\mathcal{V}_q^{\tau,r}$ in G^τ.
$\mathcal{Q}^{\tau,r}$	$\mathcal{Q}^{\tau,r}$ contains nodes $q \in \mathcal{V}^\kappa$ for which $\mathcal{V}_q^{\tau,r}$ is nonempty.

Table 4 (continued)

Symbol	Description
$f_{\mathcal{S}}^{\tau}$	Map from an index in $\mathcal{S}^{\tau}$ to the corresponding index in $\mathcal{S}$.
$f_{\mathcal{Q}}^{\tau,r}$	Map from an index in $\mathcal{Q}^{\tau,r}$ to the corresponding index in $\mathcal{V}$.
k	Index of a subgraph in $\mathcal{S}^{\tau}$.
j	Index of a node in $\mathcal{Q}^{\tau,r}$.
v'_w	The w-th node of $\mathcal{V}^{\tau}$ under the row ordering of $\mathbf{X}^{\tau}$.
Translation, views, and membership matrices	
T_v	S2N node translation function mapping a subgraph to a node.
T_e	S2N edge translation function mapping a pair of subgraphs to an edge weight.
$\widehat{G}$	Graph obtained by Subgraph-To-Node translation.
$\widehat{\mathcal{V}}$	Node set of $\widehat{G}$, with one node for each subgraph in $\mathcal{S}$.
$\widehat{\mathbf{A}}$	Weighted adjacency matrix of $\widehat{G}$.
$\hat{v}_i$	Node representing subgraph S_i in $\widehat{G}$.
$s_k^{\tau,r}$	Node representing subgraph $S^{\tau}_{f^{\tau}_{\mathcal{S}}(k)}$ after S2N translation.
$p_{q_j}^{\tau,r}$	Node representing projected subgraph $P_{q_j}^{\tau,r}$ after S2N translation.
$\widehat{G}^{\tau,r}_{\text{node}}$, $\widehat{G}^{\tau,r}_{\text{edge}}$	Node-overlap and edge-connectivity views, respectively, for node type τ and edge type r.
$\widehat{\mathcal{V}}^{\tau,r}$	Node set shared by the node-overlap and edge-connectivity views, containing nodes representing target and projected subgraphs.
$\widehat{\mathbf{X}}^{\tau,r}$	Initial node feature matrix shared by the node-overlap and edge-connectivity views.
$\widehat{\mathcal{E}}^{\tau,r}_{\text{node}}$, $\widehat{\mathcal{E}}^{\tau,r}_{\text{edge}}$	Edge sets of the node-overlap and edge-connectivity views, respectively.
$\widehat{\mathbf{A}}^{\tau,r}_{\text{node}}$, $\widehat{\mathbf{A}}^{\tau,r}_{\text{edge}}$	Weighted adjacency matrices of the node-overlap and edge-connectivity views, respectively.
$\mathbf{M}_S^{\tau,r}$, $\mathbf{M}_P^{\tau,r}$	Binary membership matrices of target and projected subgraphs, respectively; rows correspond to subgraphs, columns to nodes in $\mathcal{V}^{\tau}$, and entries indicate membership.
$\mathbf{M}^{\tau,r}$	Binary membership matrix for target and projected subgraphs, with target rows preceding projected-subgraph rows.
$\mathbf{A}^{\tau}$	$\mathbf{A}^{\tau}$ is the adjacency matrix of G^{τ}.
$\mathcal{G}_{\text{node}}$, $\mathcal{G}_{\text{edge}}$, $\mathcal{G}_{\text{same}}$	Sets of node-overlap, edge-connectivity, and same-type views, respectively; the latter two include only types whose base graphs contain edges.
$\mathcal{G}_{\text{view}}$	Set of all node-overlap, edge-connectivity, and same-type views used for message passing.
$\mathcal{V}_{G'}$	Node set of view G'.
$\mathcal{E}_{G'}$	Edge set of view G'.
$\mathbf{A}_{G'}$	Adjacency matrix of view G'.
$\mathbf{X}_{G'}$	Initial node feature matrix of view G'.
$\mathbf{d}_{\text{node}}$, $\mathbf{d}_{\text{edge}}$	Uncorrected degree vectors of the node-overlap and edge-connectivity adjacency products, respectively.
Message passing and learned representations	
ℓ	Layer index.
L	Total number of CTAP layers.
$\text{MPNN}^{(\ell)}_{G'}$	$\text{MPNN}^{(\ell)}_{G'}$ is the MPNN used on view G' at layer ℓ.
Aggregate	Function combining neighboring node features in an MPNN layer.
Update	Function mapping aggregated features to the next node representations in an MPNN layer.
$\mathbf{H}^{(\ell)}_{G'}$	Feature matrix of view G' updated by membership pooling at layer ℓ.
$\widehat{\mathbf{H}}^{(\ell)}_{G'}$	MPNN output matrix for view G' at layer ℓ.
$\mathbf{H}^{\tau,r,(\ell)}_{\text{node}}$, $\mathbf{H}^{\tau,r,(\ell)}_{\text{edge}}$, $\mathbf{H}^{\tau,(\ell)}$	Feature matrices $\mathbf{H}^{(\ell)}_{G'}$ for the node-overlap, edge-connectivity, and same-type views, respectively.

Table 4 (continued)

Symbol	Description
$\widehat{\mathbf{h}}^{\tau,r,(\ell)}_{\text{node},q}$, $\widehat{\mathbf{h}}^{\tau,r,(\ell)}_{\text{edge},q}$, $\widehat{\mathbf{h}}^{\kappa,(\ell)}_{q}$	Outputs of message passing representing node q in the node-overlap, edge-connectivity, and same-type views, respectively, at layer ℓ.
$\widehat{\mathbf{h}}^{\tau,r,(\ell)}_{\text{node},S_i}$, $\widehat{\mathbf{h}}^{\tau,r,(\ell)}_{\text{edge},S_i}$	Outputs of message passing representing subgraph S_i^τ in the node-overlap and edge-connectivity views, respectively, at layer ℓ.
$\mathcal{H}^{(\ell)}_{\text{node},q}$, $\mathcal{H}^{(\ell)}_{\text{edge},q}$, $\mathcal{H}^{(\ell)}_{\text{same},q}$	Sets of outputs of message passing representing node q across node-overlap, edge-connectivity, and same-type views, respectively, at layer ℓ.
$\mathcal{H}^{\tau,(\ell)}_{\text{node},S_i}$, $\mathcal{H}^{\tau,(\ell)}_{\text{edge},S_i}$	Sets of outputs of message passing representing subgraph S_i^τ across node-overlap and edge-connectivity views, respectively, at layer ℓ.
$\mathbf{H}^{(\ell)}_{\text{node},q}$, $\mathbf{H}^{(\ell)}_{\text{edge},q}$, $\mathbf{H}^{(\ell)}_{\text{same},q}$	Matrices whose rows are the outputs representing node q across node-overlap, edge-connectivity, and same-type views, respectively, at layer ℓ.
$\mathbf{H}^{\tau,(\ell)}_{\text{node},S_i}$, $\mathbf{H}^{\tau,(\ell)}_{\text{edge},S_i}$	Matrices whose rows are the outputs representing subgraph S_i^τ across node-overlap and edge-connectivity views, respectively, at layer ℓ.
$\mathcal{H}^{(\ell)}_{q}$, $\mathcal{H}^{\tau,(\ell)}_{S_i}$	Sets of outputs of message passing representing node q and subgraph S_i^τ, respectively, across available views at layer ℓ.
$\mathbf{H}^{(\ell)}_{q}$, $\mathbf{H}^{\tau,(\ell)}_{S_i}$	Input matrices for semantic fusion whose rows are the outputs representing node q and subgraph S_i^τ, respectively, across views at layer ℓ.
$\mathbf{u}^{(\ell)}_{q}$, $\mathbf{u}^{\tau,(\ell)}_{S_i}$	Representations of node q and subgraph S_i^τ, respectively, after semantic fusion of view outputs at layer ℓ.
Pool	Pooling function over a set of node features.
$\mathbf{c}_q^{\tau,r,(\ell)}$	Pooled representation of the neighbors of q in $\mathcal{V}_q^{\tau,r}$ at layer ℓ.
$\mathbf{C}_q^{(\ell)}$	Matrix whose columns are the pooled representations $\mathbf{c}_q^{\tau,r,(\ell)}$ for relations in $\mathcal{R}_q$.
$\mathbf{c}_{S_i}^{\tau,(\ell)}$	Pooled representation of the constituent nodes of subgraph S_i^τ at layer ℓ.
$\mathbf{h}^{(\ell)}_{q}$, $\mathbf{h}^{\tau,(\ell)}_{S_i}$	Updated representations of node q and subgraph S_i^τ, respectively, after membership pooling at layer ℓ.
$\mathbf{H}^{(L)}_{S_i}$	Matrix whose rows are the final representations of S_i^τ across node types.
$\mathbf{h}_{S_i}$	Learned representation of subgraph S_i.
Semantic fusion	
SF	Attention-based semantic fusion function combining the rows of an input matrix into a single feature vector.
$\text{SF}^{(\ell)}_{\text{view,node}}$, $\text{SF}^{(\ell)}_{\text{view,target}}$	Semantic fusion functions for view outputs representing nodes in the base graph and subgraphs S_i^τ, respectively, at layer ℓ.
$\text{SF}^{(\ell)}_{\text{member,node}}$, $\text{SF}^{(\ell)}_{\text{member,target}}$	Semantic fusion functions used in membership pooling for nodes in the base graph and subgraphs S_i^τ, respectively, at layer ℓ.
$\text{SF}_{\text{subgraph}}$	Semantic fusion function for the final representations of a target subgraph across node types.
$\mathbf{H} = [\mathbf{h}_1, \ldots, \mathbf{h}_m]^\top$	Input matrix whose rows are the feature vectors to be combined by semantic fusion.
m	Number of rows in the input matrix $\mathbf{H}$, including padded positions when needed.
$\mathbf{Q}$, $\mathbf{K}$, $\mathbf{V}$	Query, key, and value matrices in semantic fusion, respectively.
d_k	Dimension of the query and key vectors.
$\mathbf{W}_Q$, $\mathbf{W}_K$, $\mathbf{W}_V$	Trainable query, key, and value projection matrices in semantic fusion, respectively.
$\mathbf{W}_O$	Trainable output projection matrix in semantic fusion.
$\mathbf{P}$	Attention matrix in semantic fusion.
γ	Trainable residual coefficient in semantic fusion.
b	Batch index for semantic fusion.
B	Number of elements in a batch for semantic fusion.
$\mathbf{H}_b$	Tensor containing the input matrices for batch b in semantic fusion.
$\mathbf{h}_{b,i,j}$	Feature vector at source position j of element i in batch b.
$\mathbf{Q}_b$, $\mathbf{K}_b$, $\mathbf{V}_b$	Query, key, and value tensors for batch b in semantic fusion, respectively.
$\mathbf{P}_b$	Attention tensor for batch b in semantic fusion.

Table 4 (continued)

Symbol	Description
$\mathbf{u}_b$	Matrix of semantic fusion outputs for batch b.
vec	Operator concatenating matrix rows into a vector.
softmax	Softmax function over source positions in attention.
Classification, complexity, and general operators	
$\mathcal{Y}$	Collection of class labels for the subgraphs in $\mathcal{S}$.
$\mathbf{y}_{S_i}$	Ground-truth label vector for subgraph S_i.
d_y	Number of classes.
f_{cls}	Classification head for subgraph representations.
$\hat{\mathbf{y}}_{S_i}$	Logits vector for subgraph S_i, with one unnormalized score per class.
ℓ_{cls}	Classification loss function for one subgraph.
$\mathcal{L}$	Training loss over the subgraphs in $\mathcal{S}_{\text{train}}$.
$\mathcal{L}_B$	Training loss for minibatch $\mathcal{S}_B$.
$c_v^{\tau,r}$	$c_v^{\tau,r}$ is the number of nonzero entries in the column for node v of $\mathbf{M}^{\tau,r}$.
T_{MPNN}	T_{MPNN} denotes the MPNN computation cost per CTAP layer in the implementation.
$O(\cdot)$	$O(\cdot)$ denotes asymptotic computational complexity.
$\mathbf{1}$	Vector of ones.
$\mathbf{1}[\cdot]$	Indicator function, equal to one when its condition holds and zero otherwise.
$\lvert \cdot \rvert$	Set cardinality.
$(\cdot)^\top$	Matrix or vector transpose.
$[\cdot, \cdot]$	Column concatenation of vectors or matrices.
$(\cdot)_{i,:}$	Row i of a matrix.
$(\cdot)_{:,j}$	Column j of a matrix.

B Additional Considerations for Subgraph Representation Learning on Heterogeneous Graphs

Changing the base graph from homogeneous to heterogeneous introduces additional considerations for representing a target subgraph: the information associated with each node type, the relationship between a type-specific target node set and nodes of other types, and dependencies mediated by those types. We explain these considerations and how CTAP addresses them below.

B.1 Information Associated with Different Node Types

Within a target subgraph, the nodes of different types can describe different aspects of the same object. A representation should preserve what each type-specific node set conveys before combining these aspects. Accounting for types in individual node embeddings does not explicitly provide a representation of each such set within the target.

Type-aware heterogeneous graph pooling provides a precedent for separately representing information associated with different types and then combining it (Wu et al., 2021; Ji et al., 2023; Hong et al., 2026). We extend this principle to subgraph representation learning: our hypothesis is that the type-specific node sets of a target can likewise carry distinct information worth representing separately. Unlike an entire graph, a target subgraph is defined within a shared base graph, so each set must be represented in relation to the surrounding graph. CTAP therefore decomposes both the target and the base graph by node type, learns a target representation within each resulting subproblem, and fuses these representations after the final layer.

B.2 Relations Between a Type-Specific Target Node Set and Nodes of Other Types

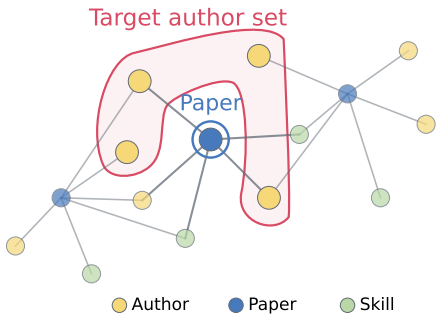

Figure 7: An author set and a related paper node.

For each type-specific node set within a target, we need to capture the relation between the subgraph it represents and an actual node of another type. Figure 7 illustrates this in an author–paper–skill network. The shaded region marks the author set within a target, and the highlighted paper is connected to two of its members and an author outside the set. Other papers connect to different subsets of this author set. The relation to a paper therefore describes which part of this author set participated in that collaboration, rather than requiring it to coincide with the paper's complete author set. Across targets, the same paper provides a common reference for expressing these set-level relations.

CTAP projects a node of another type into a subgraph defined by its neighbors, allowing the type-specific target set and that node to be compared within the same type-specific base graph. Node-overlap and edge-connectivity views express this relationship through shared membership and connections between their node sets.

B.3 Dependencies Mediated by Other Node Types

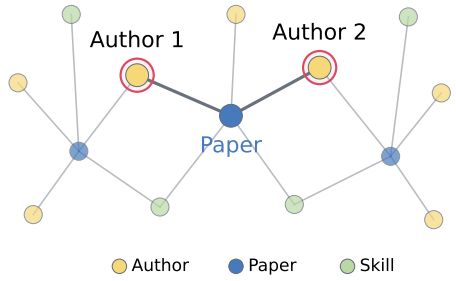

Figure 8: An author–author relation mediated by a paper.

Dependencies among the nodes belonging to a target already matter in homogeneous subgraph representation learning. GPEN and ISNN address distant-node relationships and long-range dependencies in this setting (Wu et al., 2025; Zhong et al., 2025). In a heterogeneous graph, nodes of another type can intervene between related nodes. Figure 8 shows a minimal example: two authors are connected through a paper, so their collaboration is represented by a two-hop path rather than a direct author–author edge. The example adds just one intermediate node; dependencies mediated by additional nodes require information to travel farther. Consequently, relevant dependencies can become more indirect, increasing the burden on long-range dependency modeling.

CTAP addresses these indirect dependencies by representing targets and projected anchors as nodes in translated views. Their representations exchange information directly through shared membership and connections between their node sets, reducing reliance on successive node-level updates along paths in the original graph.

C Implementation Details

C.1 Formal Definition of Type-Aware Decomposition

For each node type $\tau \in \mathcal{T}_{\mathcal{V}}$, let $\mathcal{V}^{\tau} = \{v \in \mathcal{V} \mid f_{\mathcal{V}}(v) = \tau\}$ and $\mathcal{V}^{\tau}_{S_i} = \{v \in \mathcal{V}_{S_i} \mid f_{\mathcal{V}}(v) = \tau\}$. In the corresponding subproblem, G^{τ} is given as a base graph and each S_i^{τ} is given as a subgraph. Here, $\mathcal{V}^{\tau}$ and $\mathcal{V}^{\tau}_{S_i}$ induce $\mathcal{E}^{\tau} = \{\{u, v\} \in \mathcal{E} \mid u, v \in \mathcal{V}^{\tau}\}$ and $\mathcal{E}^{\tau}_{S_i} = \{\{u, v\} \in \mathcal{E}_{S_i} \mid u, v \in \mathcal{V}^{\tau}_{S_i}\}$, respectively. Let $\mathbf{X}^{\tau}$ and $\mathbf{X}^{\tau}_{S_i}$ contain the rows of $\mathbf{X}$ corresponding to $\mathcal{V}^{\tau}$ and $\mathcal{V}^{\tau}_{S_i}$, respectively. The resulting node-induced subgraphs are

$$G^{\tau} = (\mathcal{V}^{\tau}, \mathcal{E}^{\tau}, \mathbf{X}^{\tau}), \quad S_i^{\tau} = (\mathcal{V}^{\tau}_{S_i}, \mathcal{E}^{\tau}_{S_i}, \mathbf{X}^{\tau}_{S_i}). \tag{9}$$

If $\mathcal{V}^{\tau}_{S_i} = \emptyset$, we exclude that sample from this subproblem. The remaining targets form $\mathcal{S}^{\tau} = \{S_i^{\tau} \mid S_i \in \mathcal{S},\ \mathcal{V}^{\tau}_{S_i} \neq \emptyset\}$. This subproblem is itself a subgraph representation learning problem.

C.2 Construction of the Node-Overlap View

For a node type τ, let G^τ be the base graph induced by the nodes of that type, and let S_i^τ be the corresponding part of target subgraph S_i. Let $\kappa \neq \tau$ be another node type and r an edge type connecting nodes of types τ and κ. For a κ-type node q, $\mathcal{V}_q^{\tau,r}$ contains its τ-type neighbors connected through edges of type r, and $P_q^{\tau,r}$ is the subgraph induced by this set in G^τ. These are the type-specific graphs and cross-type projections defined in Section 4.1.

We use the target set $\mathcal{S}^\tau$ defined in Section 4.1 and collect the nodes retained for this view:

$$\mathcal{Q}^{\tau,r} = \{q \in \mathcal{V}^\kappa \mid \mathcal{V}_q^{\tau,r} \neq \emptyset\}. \tag{10}$$

We index the samples in $\mathcal{S}^\tau$ by k and the nodes in $\mathcal{Q}^{\tau,r}$ by j. The function $f_{\mathcal{S}}^\tau(\cdot)$ maps index k in $\mathcal{S}^\tau$ to the corresponding index in $\mathcal{S}$. The function $f_{\mathcal{Q}}^{\tau,r}(\cdot)$ maps index j in $\mathcal{Q}^{\tau,r}$ to the corresponding index in $\mathcal{V}$. For $1 \leq k \leq |\mathcal{S}^\tau|$ and $q_j \in \mathcal{Q}^{\tau,r}$, we use the following mapping for the S2N node translation function T_v defined in Appendix C.15:

$$T_v(S^\tau_{f^\tau_{\mathcal{S}}(k)}) = s_k^{\tau,r}, \quad T_v(P_{q_j}^{\tau,r}) = p_{q_j}^{\tau,r}. \tag{11}$$

The node set of the view is

$$\widehat{\mathcal{V}}^{\tau,r} = \{s_k^{\tau,r} \mid 1 \leq k \leq |\mathcal{S}^\tau|\} \cup \{p_{q_j}^{\tau,r} \mid q_j \in \mathcal{Q}^{\tau,r}\}. \tag{12}$$

For any two subgraphs S and S' with node sets $\mathcal{V}_S$ and $\mathcal{V}_{S'}$, we use node-overlap counts for the S2N edge translation function T_e:

$$T_e(S, S') = |\mathcal{V}_S \cap \mathcal{V}_{S'}|. \tag{13}$$

We introduce a binary membership matrix $\mathbf{M}^{\tau,r}$ whose rows correspond to target and projected subgraphs and whose columns correspond to nodes in $\mathcal{V}^\tau$; each entry indicates whether the node belongs to the subgraph. Its construction is detailed in Appendix C.3. The weighted adjacency matrix is then obtained by

$$\widehat{\mathbf{A}}^{\tau,r}_{\text{node}} = \mathbf{M}^{\tau,r}(\mathbf{M}^{\tau,r})^\top. \tag{14}$$

The edge set is given by the nonzero entries of this matrix:

$$\widehat{\mathcal{E}}^{\tau,r}_{\text{node}} = \{\{u, v\} \mid u, v \in \widehat{\mathcal{V}}^{\tau,r},\ (\widehat{\mathbf{A}}^{\tau,r}_{\text{node}})_{ij} > 0\}, \tag{15}$$

where i and j are the indices of nodes u and v, respectively, under the node ordering used in $\mathbf{M}^{\tau,r}$. Each edge retains its weight from $\widehat{\mathbf{A}}^{\tau,r}_{\text{node}}$. We define the node feature matrix $\widehat{\mathbf{X}}^{\tau,r} \in \mathbb{R}^{|\widehat{\mathcal{V}}^{\tau,r}| \times d}$ under the same node ordering. A target-subgraph node uses the sum of the given initial features of its member nodes, while a projected-subgraph node uses the given initial feature $\mathbf{X}_{f_{\mathcal{Q}}^{\tau,r}(j)}$ of the original node q_j:

$$(\widehat{\mathbf{X}}^{\tau,r})_k = \sum_{w=1}^{|\mathcal{V}^\tau|} (\mathbf{M}^{\tau,r})_{kw}(\mathbf{X}^\tau)_w, \quad (\widehat{\mathbf{X}}^{\tau,r})_{|\mathcal{S}^\tau|+j} = \mathbf{X}_{f_{\mathcal{Q}}^{\tau,r}(j)}, \tag{16}$$

where $1 \leq k \leq |\mathcal{S}^\tau|$ and $1 \leq j \leq |\mathcal{Q}^{\tau,r}|$.

C.3 Construction of the Membership Matrix

To compute the view, we form membership matrices for the target and projected subgraphs separately. The matrices $\mathbf{M}_S^{\tau,r} \in \{0,1\}^{|\mathcal{S}^\tau| \times |\mathcal{V}^\tau|}$ and $\mathbf{M}_P^{\tau,r} \in \{0,1\}^{|\mathcal{Q}^{\tau,r}| \times |\mathcal{V}^\tau|}$ use the local sample index k and node index j for their respective rows, with entries

$$(\mathbf{M}_S^{\tau,r})_{kw} = \begin{cases} 1, & v'_w \in \mathcal{V}^\tau_{S_{f^\tau_{\mathcal{S}}(k)}}, \\ 0, & v'_w \notin \mathcal{V}^\tau_{S_{f^\tau_{\mathcal{S}}(k)}}, \end{cases} \quad (\mathbf{M}_P^{\tau,r})_{jw} = \begin{cases} 1, & v'_w \in \mathcal{V}^{\tau,r}_{q_j}, \\ 0, & v'_w \notin \mathcal{V}^{\tau,r}_{q_j} \end{cases}, \tag{17}$$

where v'_w is the w-th node of $\mathcal{V}^\tau$ under the row ordering of $\mathbf{X}^\tau$, $1 \leq w \leq |\mathcal{V}^\tau|$. We concatenate these matrices vertically, ordering the target-subgraph nodes before the projected-subgraph nodes:

$$\mathbf{M}^{\tau,r} = \begin{pmatrix} \mathbf{M}_S^{\tau,r} \\ \mathbf{M}_P^{\tau,r} \end{pmatrix}. \tag{18}$$

Thus, $s_k^{\tau,r}$ occupies row k and $p_{q_j}^{\tau,r}$ occupies row $|\mathcal{S}^\tau| + j$.

C.4 Construction of the Edge-Connectivity View

For a node type τ, let G^τ be the base graph induced by the nodes of that type, and assume that G^τ contains edges. Let $\kappa \neq \tau$ be another node type and r an edge type connecting nodes of types τ and κ. We use the target subgraphs S_i^τ and the projected subgraphs $P_q^{\tau,r}$ induced by the τ-type neighbors of κ-type nodes q through relation r, as in Appendix C.2.

The edge-connectivity view uses the same mapping for T_v as Equation (11), and its node set is $\widehat{\mathcal{V}}^{\tau,r}$ in Equation (12). For any two subgraphs S and S' in G^τ with node sets $\mathcal{V}_S$ and $\mathcal{V}_{S'}$, we instead use the sum of adjacency entries between their respective nodes for T_e:

$$T_e(S, S') = \sum_{u \in \mathcal{V}_S} \sum_{v \in \mathcal{V}_{S'}} \mathbf{1}[\{u, v\} \in \mathcal{E}^\tau]. \tag{19}$$

An edge whose two distinct endpoints belong to both subgraphs contributes twice to this weight.

Using the membership matrix $\mathbf{M}^{\tau,r}$ in Equation (18), we obtain the weighted adjacency matrix as

$$\widehat{\mathbf{A}}^{\tau,r}_{\text{edge}} = \mathbf{M}^{\tau,r} \mathbf{A}^\tau (\mathbf{M}^{\tau,r})^\top, \tag{20}$$

where $\mathbf{A}^\tau \in \{0,1\}^{|\mathcal{V}^\tau| \times |\mathcal{V}^\tau|}$ is the adjacency matrix of G^τ, with its rows and columns ordered as the columns of $\mathbf{M}^{\tau,r}$. Following Equation (15), we define the edge set by replacing $\widehat{\mathbf{A}}^{\tau,r}_{\text{node}}$ with $\widehat{\mathbf{A}}^{\tau,r}_{\text{edge}}$:

$$\widehat{\mathcal{E}}^{\tau,r}_{\text{edge}} = \{\{u, v\} \mid u, v \in \widehat{\mathcal{V}}^{\tau,r},\ (\widehat{\mathbf{A}}^{\tau,r}_{\text{edge}})_{ij} > 0\}, \tag{21}$$

where i and j are the indices of nodes u and v, respectively, under the node ordering used in $\mathbf{M}^{\tau,r}$. Each edge retains its weight from $\widehat{\mathbf{A}}^{\tau,r}_{\text{edge}}$. The initial node feature matrix $\widehat{\mathbf{X}}^{\tau,r}$ is shared with the corresponding node-overlap view.

C.5 Views for Message Passing

We define the three collections of views used for message passing as follows:

$$\mathcal{G}_{\text{node}} = \{\widehat{G}^{\tau,r}_{\text{node}} \mid \tau \in \mathcal{T}_\mathcal{V},\ r \in \mathcal{C}^\tau\}. \tag{22}$$

$$\mathcal{G}_{\text{edge}} = \{\widehat{G}^{\tau,r}_{\text{edge}} \mid \tau \in \mathcal{T}_\mathcal{V},\ r \in \mathcal{C}^\tau,\ \mathcal{E}^\tau \neq \emptyset\}. \tag{23}$$

$$\mathcal{G}_{\text{same}} = \{G^\tau \mid \tau \in \mathcal{T}_\mathcal{V},\ \mathcal{E}^\tau \neq \emptyset\}. \tag{24}$$

Together, they form $\mathcal{G}_{\text{view}} = \mathcal{G}_{\text{node}} \cup \mathcal{G}_{\text{edge}} \cup \mathcal{G}_{\text{same}}$.

For $G' = \widehat{G}^{\tau,r}_{\text{node}}$, $(\mathbf{A}_{G'}, \mathbf{X}_{G'}, \mathbf{H}^{(\ell-1)}_{G'}, \widehat{\mathbf{H}}^{(\ell)}_{G'})$ corresponds to $(\widehat{\mathbf{A}}^{\tau,r}_{\text{node}}, \widehat{\mathbf{X}}^{\tau,r}, \mathbf{H}^{\tau,r,(\ell-1)}_{\text{node}}, \widehat{\mathbf{H}}^{\tau,r,(\ell)}_{\text{node}})$. For $G' = \widehat{G}^{\tau,r}_{\text{edge}}$, it corresponds to $(\widehat{\mathbf{A}}^{\tau,r}_{\text{edge}}, \widehat{\mathbf{X}}^{\tau,r}, \mathbf{H}^{\tau,r,(\ell-1)}_{\text{edge}}, \widehat{\mathbf{H}}^{\tau,r,(\ell)}_{\text{edge}})$. For $G' = G^\tau$, it corresponds to $(\mathbf{A}^\tau, \mathbf{X}^\tau, \mathbf{H}^{\tau,(\ell-1)}, \widehat{\mathbf{H}}^{\tau,(\ell)})$. In these input matrices, the corresponding target or node features are stored in rows k and $|\mathcal{S}^\tau| + j$ for translated views and row j' for same-type views, as follows:

$$(\mathbf{H}^{\tau,r,(\ell)}_{\text{node}})_k = (\mathbf{H}^{\tau,r,(\ell)}_{\text{edge}})_k = \mathbf{h}^{\tau,(\ell)}_{S_i}, \qquad f^\tau_\mathcal{S}(k) = i. \tag{25}$$

$$(\mathbf{H}^{\tau,r,(\ell)}_{\text{node}})_{|\mathcal{S}^\tau|+j} = (\mathbf{H}^{\tau,r,(\ell)}_{\text{edge}})_{|\mathcal{S}^\tau|+j} = \mathbf{h}^{(\ell)}_q, \qquad q = v_{f^{\tau,r}_\mathcal{Q}(j)}. \tag{26}$$

$$(\mathbf{H}^{\kappa,(\ell)})_{j'} = \mathbf{h}^{(\ell)}_q, \qquad q = v'_{j'}, \tag{27}$$

where $v'_{j'}$ follows the row ordering of $\mathbf{X}^\kappa$, whenever the same-type view exists. The hat distinguishes the MPNN output from the feature matrix updated by membership pooling for the next layer. Each $\text{MPNN}^{(\ell)}_{G'}$ has separate parameters.

C.6 Detailed Structure of an MPNN Layer

For a view with adjacency $\mathbf{A}$ and input features $\mathbf{H}$, we express an MPNN layer as

$$\mathrm{MPNN}(\mathbf{A}, \mathbf{H}) = \mathrm{Update}(\mathrm{Aggregate}(\mathbf{A}, \mathbf{H})).$$

Here, $\mathrm{Aggregate}(\cdot, \cdot)$ combines the neighboring node features, and $\mathrm{Update}(\cdot)$ maps the resulting features to the next node representations. For weighted sums, $\mathrm{Aggregate}(\mathbf{A}, \mathbf{H}) = \mathbf{AH}$. The node-overlap and edge-connectivity views satisfy

$$\widehat{\mathbf{A}}^{\tau,r}_{\mathrm{node}} \mathbf{H}^{\tau,r,(\ell-1)}_{\mathrm{node}} = \mathbf{M}^{\tau,r}((\mathbf{M}^{\tau,r})^\top \mathbf{H}^{\tau,r,(\ell-1)}_{\mathrm{node}}),$$
$$\widehat{\mathbf{A}}^{\tau,r}_{\mathrm{edge}} \mathbf{H}^{\tau,r,(\ell-1)}_{\mathrm{edge}} = \mathbf{M}^{\tau,r}(\mathbf{A}^{\tau}((\mathbf{M}^{\tau,r})^\top \mathbf{H}^{\tau,r,(\ell-1)}_{\mathrm{edge}})),$$

respectively. Constructing the translated adjacency matrices and multiplying them by the features evaluates the left-hand sides. However, these matrices can contain many edges when subgraphs overlap or are connected. To avoid processing these translated edges, our implementation evaluates the right-hand sides using $\mathbf{M}^{\tau,r}$ and $\mathbf{A}^{\tau}$ without first combining them into translated adjacency matrices. The identities preserve the weighted sums for the translated views. The same-type views compute $\mathbf{A}^{\tau}\mathbf{H}^{\tau,(\ell-1)}$.

For the left-hand sides, the costs per layer are $O(|\widehat{\mathcal{E}}^{\tau,r}_{\mathrm{node}}|d)$ and $O(|\widehat{\mathcal{E}}^{\tau,r}_{\mathrm{edge}}|d)$ for the node-overlap and edge-connectivity views, respectively, where d is the feature dimension. In our implementation, the right-hand sides cost $O((\sum_{S_i^\tau \in \mathcal{S}^\tau} |\mathcal{V}^\tau_{S_i}| + \sum_{q \in \mathcal{Q}^{\tau,r}} |\mathcal{V}^{\tau,r}_q|)d)$ and $O((\sum_{S_i^\tau \in \mathcal{S}^\tau} |\mathcal{V}^\tau_{S_i}| + \sum_{q \in \mathcal{Q}^{\tau,r}} |\mathcal{V}^{\tau,r}_q| + |\mathcal{E}^\tau|)d)$, in the same order. Appendix D.3 derives these costs.

Our implementation thus replaces propagation over translated edges with products using memberships and base edges. Appendix E.4 quantifies this reduction on the six datasets. The matrices $\mathbf{M}^{\tau,r}$ and $\mathbf{A}^{\tau}$ are reused, while the products with the features are evaluated at each layer.

Next, $\mathrm{Update}(\cdot)$ transforms each node's aggregated features into its next representation using a dense linear transformation or an MLP composed of dense linear transformations and activation functions. The transformation parameters are shared across nodes within the same view. The cost of this update is derived in Appendix D.3.

C.7 Architecture Implementation

Each view encoder consists of stacked MPNN layers. In our experiments, each nonfinal layer uses a GCN (Kipf & Welling, 2017) convolution followed by GraphNorm (Cai et al., 2021) for normalization, ELU as the activation function, and dropout. The final layer applies only the convolution. Each view has its own encoder parameters. The translated adjacency matrices use the normalization defined by GCN and unit self-loops; for the same-type views, we use PyTorch Geometric's `GCNConv` (Fey & Lenssen, 2019).

GCN normalization likewise avoids translated adjacencies: the uncorrected degree vectors are $\mathbf{d}_{\mathrm{node}} = \mathbf{M}(\mathbf{M}^\top \mathbf{1})$ and $\mathbf{d}_{\mathrm{edge}} = \mathbf{M}(\mathbf{A}^{\tau}(\mathbf{M}^\top \mathbf{1}))$, with the product diagonal subtracted and unit self-loops added before inverse-square-root normalization, where $\mathbf{M} = \mathbf{M}^{\tau,r}$.

For semantic fusion, we use a self-attention module of the same form as the Transformer-based semantic fusion in SeHGNN (Yang et al., 2023). The module computes attention across the input feature vectors, combines the attended features with the inputs through a learned residual coefficient, and concatenates the resulting vectors before a linear projection. For feature width d, the projections for queries and keys have width $d/4$, while the projection for values has width d. The residual coefficient γ is initialized to zero.

C.8 Construction of Input Matrices $\mathbf{H}^{(\ell)}_q$, $\mathbf{H}^{\tau,(\ell)}_{S_i}$, $\mathbf{C}^{(\ell)}_q$, and $\mathbf{H}^{(L)}_{S_i}$ for Semantic Fusion

This subsection describes how the vectors used in Sections 5.2–5.4 are arranged into matrices for semantic fusion, including $\mathbf{H}^{(\ell)}_q$, $\mathbf{H}^{\tau,(\ell)}_{S_i}$, $\mathbf{C}^{(\ell)}_q$, and $\mathbf{H}^{(L)}_{S_i}$. The matrices $\widehat{\mathbf{H}}^{(\ell)}_{\widehat{G}^{\tau,r}_{\mathrm{node}}}$, $\widehat{\mathbf{H}}^{(\ell)}_{\widehat{G}^{\tau,r}_{\mathrm{edge}}}$, and $\widehat{\mathbf{H}}^{(\ell)}_{G^\tau}$ contain the outputs of message passing at layer ℓ on node-overlap, edge-connectivity, and same-type views, respectively, with the row ordering of their initial feature matrices.

The target set $\mathcal{S}^\tau$ is defined in Section 4.1, and the index maps $f^\tau_{\mathcal{S}}(\cdot)$ and $f^{\tau,r}_{\mathcal{Q}}(\cdot)$ are defined in Appendix C.2. For each node $q = v_i \in \mathcal{V}$, let $\kappa = f_{\mathcal{V}}(q)$. The cross-type relations incident to q are

$$\mathcal{R}_q = \{f_{\mathcal{E}}(\{v, q\}) \mid \{v, q\} \in \mathcal{E},\ f_{\mathcal{V}}(v) \neq \kappa\}. \tag{28}$$

For each $r \in \mathcal{R}_q$, let τ be the node type at the other end of relation r from κ, and let j satisfy $f^{\tau,r}_{\mathcal{Q}}(j) = i$. We denote the corresponding output features as $\widehat{\mathbf{h}}^{\tau,r,(\ell)}_{\mathrm{node},q} := (\widehat{\mathbf{H}}^{(\ell)}_{\widehat{G}^{\tau,r}_{\mathrm{node}}})_{|\mathcal{S}^\tau|+j}$, $\widehat{\mathbf{h}}^{\tau,r,(\ell)}_{\mathrm{edge},q} := (\widehat{\mathbf{H}}^{(\ell)}_{\widehat{G}^{\tau,r}_{\mathrm{edge}}})_{|\mathcal{S}^\tau|+j}$, if the respective views exist. For the same-type view G^κ, let $q = v'_{j'}$ in the node ordering of $\mathcal{V}^\kappa$ used by $\mathbf{X}^\kappa$. We denote its output feature as $\widehat{\mathbf{h}}^{\kappa,(\ell)}_q := (\widehat{\mathbf{H}}^{(\ell)}_{G^\kappa})_{j'}$, if the same-type view exists.

The view outputs for node q form the sets

$$\mathcal{H}^{(\ell)}_{\mathrm{node},q} = \{\widehat{\mathbf{h}}^{\tau,r,(\ell)}_{\mathrm{node},q} \mid r \in \mathcal{R}_q\}, \tag{29}$$

$$\mathcal{H}^{(\ell)}_{\mathrm{edge},q} = \{\widehat{\mathbf{h}}^{\tau,r,(\ell)}_{\mathrm{edge},q} \mid r \in \mathcal{R}_q,\ \mathcal{E}^\tau \neq \emptyset\}, \tag{30}$$

$$\mathcal{H}^{(\ell)}_{\mathrm{same},q} = \{\widehat{\mathbf{h}}^{\kappa,(\ell)}_q \mid \mathcal{E}^\kappa \neq \emptyset\}, \tag{31}$$

with $\mathcal{H}^{(\ell)}_q = \mathcal{H}^{(\ell)}_{\mathrm{node},q} \cup \mathcal{H}^{(\ell)}_{\mathrm{edge},q} \cup \mathcal{H}^{(\ell)}_{\mathrm{same},q}$.

For each target subgraph $S_i \in \mathcal{S}$ and node type τ with $S^\tau_i \in \mathcal{S}^\tau$, we also fuse the outputs of its translated nodes across views. For each $r \in \mathcal{C}^\tau$, we denote these outputs as $\widehat{\mathbf{h}}^{\tau,r,(\ell)}_{\mathrm{node},S_i} := (\widehat{\mathbf{H}}^{(\ell)}_{\widehat{G}^{\tau,r}_{\mathrm{node}}})_k$, $\widehat{\mathbf{h}}^{\tau,r,(\ell)}_{\mathrm{edge},S_i} := (\widehat{\mathbf{H}}^{(\ell)}_{\widehat{G}^{\tau,r}_{\mathrm{edge}}})_k$, where k satisfies $f^\tau_{\mathcal{S}}(k) = i$, if the respective views exist. These outputs form the sets

$$\mathcal{H}^{\tau,(\ell)}_{\mathrm{node},S_i} = \{\widehat{\mathbf{h}}^{\tau,r,(\ell)}_{\mathrm{node},S_i} \mid r \in \mathcal{C}^\tau\}, \tag{32}$$

$$\mathcal{H}^{\tau,(\ell)}_{\mathrm{edge},S_i} = \{\widehat{\mathbf{h}}^{\tau,r,(\ell)}_{\mathrm{edge},S_i} \mid r \in \mathcal{C}^\tau,\ \mathcal{E}^\tau \neq \emptyset\}, \tag{33}$$

with $\mathcal{H}^{\tau,(\ell)}_{S_i} = \mathcal{H}^{\tau,(\ell)}_{\mathrm{node},S_i} \cup \mathcal{H}^{\tau,(\ell)}_{\mathrm{edge},S_i}$.

We stack the vectors in each of the sets above as rows to form the corresponding matrices, retaining the same subscripts and superscripts and replacing $\mathcal{H}$ by $\mathbf{H}$. Within each view group, rows follow the view construction order. The complete input matrices are formed by stacking these matrices vertically:

$$\mathbf{H}^{(\ell)}_q = \begin{bmatrix} \mathbf{H}^{(\ell)}_{\mathrm{node},q} \\ \mathbf{H}^{(\ell)}_{\mathrm{edge},q} \\ \mathbf{H}^{(\ell)}_{\mathrm{same},q} \end{bmatrix}, \qquad \mathbf{H}^{\tau,(\ell)}_{S_i} = \begin{bmatrix} \mathbf{H}^{\tau,(\ell)}_{\mathrm{node},S_i} \\ \mathbf{H}^{\tau,(\ell)}_{\mathrm{edge},S_i} \end{bmatrix}. \tag{34}$$

Thus, node-overlap rows precede edge-connectivity rows, followed by the same-type row for actual nodes.

For membership pooling, let r_j be the j-th relation in $\mathcal{R}_q$ under the corresponding view construction order, and let τ_j be the node type at its other end. We define $\mathbf{C}^{(\ell)}_q \in \mathbb{R}^{d \times |\mathcal{R}_q|}$ by its columns:

$$(\mathbf{C}^{(\ell)}_q)_{:,j} = \mathbf{c}^{\tau_j, r_j, (\ell)}_q, \qquad 1 \leq j \leq |\mathcal{R}_q|, \tag{35}$$

where $(\mathbf{C}^{(\ell)}_q)_{:,j}$ denotes its j-th column. The input to $\mathrm{SF}^{(\ell)}_{\mathrm{member,node}}$ is $[\mathbf{u}^{(\ell)}_q, \mathbf{C}^{(\ell)}_q]^\top$. For $\mathrm{SF}^{(\ell)}_{\mathrm{member,target}}$, $\mathbf{u}^{\tau,(\ell)}_{S_i}$ precedes $\mathbf{c}^{\tau,(\ell)}_{S_i}$.

For fusion across node types, let τ_j be the j-th type in $\mathcal{T}_{\mathcal{V}}$ in ascending type-ID order. The matrix $\mathbf{H}^{(L)}_{S_i} \in \mathbb{R}^{|\mathcal{T}_{\mathcal{V}}| \times d}$ has the row correspondence

$$(\mathbf{H}^{(L)}_{S_i})_{j,:} = (\mathbf{h}^{\tau_j,(L)}_{S_i})^\top, \qquad \mathcal{V}^{\tau_j}_{S_i} \neq \emptyset, \tag{36}$$

where $(\mathbf{H}^{(L)}_{S_i})_{j,:}$ denotes its j-th row. Rows for types absent from S_i are zero-padded and masked. For Jumping Knowledge concatenation, the same row construction is applied at each layer $\ell = 1, \ldots, L$, as detailed in Appendix C.13.

Unavailable inputs retain their assigned positions with zero-padding and masking, as described in Appendix C.10; subsequent inputs are not shifted.

C.9 Computation of Semantic Fusion

Semantic fusion takes an input matrix $\mathbf{H} \in \mathbb{R}^{m \times d}$ whose rows are the feature vectors to be combined. The row order is specified in Appendix C.8:

$$\mathbf{H} := [\mathbf{h}_1, \ldots, \mathbf{h}_m]^\top, \tag{37}$$

$$\mathbf{Q} = \mathbf{H}\mathbf{W}_Q, \quad \mathbf{K} = \mathbf{H}\mathbf{W}_K, \quad \mathbf{V} = \mathbf{H}\mathbf{W}_V, \tag{38}$$

$$\mathbf{P} = \mathrm{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right), \tag{39}$$

$$\mathrm{SF}(\mathbf{H}) = \mathbf{W}_O \,\mathrm{vec}\big(\mathbf{H} + \gamma \mathbf{P}\mathbf{V}\big), \tag{40}$$

where d_k is the dimension of the query and key vectors. Our implementation uses $d_k = d/4$, as described in Appendix C.7. The matrices $\mathbf{W}_Q$, $\mathbf{W}_K$, $\mathbf{W}_V$, and $\mathbf{W}_O$ and the scalar γ are trainable; $\mathrm{vec}(\cdot)$ concatenates the matrix rows into a single column vector. It preserves this row order, so changing the stacking order can change the SF output. The computation of semantic fusion for batched inputs is detailed in Appendix C.10.

C.10 Batch Input to Semantic Fusion

Appendix C.9 describes semantic fusion for a single input matrix $\mathbf{H}$, as in Equation 37. Stacking these matrices along a batch axis gives the input tensor for batch b, $\mathbf{H}_b \in \mathbb{R}^{B \times m \times d}$, where b identifies the batch, B is its number of elements, m is the number of rows in each input matrix $\mathbf{H}$ (including padded positions when needed), and d is the feature width. Each slice $\mathbf{H}_b[i, :, :]$ is the matrix $\mathbf{H}$ for the i-th node in the batch:

$$\mathbf{H}_b[i, :, :] = [\mathbf{h}_{b,i,1}, \ldots, \mathbf{h}_{b,i,m}]^\top, \qquad 1 \le i \le B,$$

where $\mathbf{h}_{b,i,j}$ is the vector at source position j of element i in batch b. Applying the projections along the feature axis gives

$$\mathbf{Q}_b = \mathbf{H}_b\mathbf{W}_Q, \qquad \mathbf{K}_b = \mathbf{H}_b\mathbf{W}_K, \qquad \mathbf{V}_b = \mathbf{H}_b\mathbf{W}_V.$$

The tensors $\mathbf{Q}_b$ and $\mathbf{K}_b$ have dimensions $B \times m \times d_k$, and $\mathbf{V}_b$ has dimensions $B \times m \times d$. We compute attention as

$$\mathbf{P}_b = \mathrm{softmax}\left(\frac{\mathbf{Q}_b\mathbf{K}_b^\top}{\sqrt{d_k}}\right).$$

Here, the transpose exchanges only the last two axes, matrix multiplication is performed separately for each batch element, and $\mathrm{softmax}(\cdot)$ acts along the last axis. Thus, $\mathbf{P}_b \in \mathbb{R}^{B \times m \times m}$. The fused outputs for the batch are

$$\mathbf{u}_b = \mathrm{vec}(\mathbf{H}_b + \gamma \mathbf{P}_b\mathbf{V}_b)\mathbf{W}_O^\top.$$

Here, $\mathrm{vec}(\cdot)$ flattens the last two axes of each batch element into a row, producing a $B \times (md)$ matrix. Each row $\mathbf{u}_b[i, :]$ is the transpose of the single-input output $\mathrm{SF}(\mathbf{H})$ for the i-th node. When a vector is unavailable, its fixed input position is filled with a zero vector; the remaining vectors retain their positions. We mask the attention weights for padded positions and renormalize over the available input vectors. Padded positions are also masked after the residual combination, before applying $\mathrm{vec}(\cdot)$ and the projection that produces the output.

C.11 Subgraph Sets for Construction of Views

Let $\mathcal{S}_{\mathrm{train}}$, $\mathcal{S}_{\mathrm{val}}$, and $\mathcal{S}_{\mathrm{test}}$ denote the training, validation, and test subgraph sets. For each split, its type-specific targets are

$$\mathcal{S}^\tau_{\mathrm{train}} = \{S_i^\tau \mid S_i \in \mathcal{S}_{\mathrm{train}},\ \mathcal{V}^\tau_{S_i} \neq \emptyset\}.$$
$$\mathcal{S}^\tau_{\mathrm{val}} = \{S_i^\tau \mid S_i \in \mathcal{S}_{\mathrm{val}},\ \mathcal{V}^\tau_{S_i} \neq \emptyset\}.$$
$$\mathcal{S}^\tau_{\mathrm{test}} = \{S_i^\tau \mid S_i \in \mathcal{S}_{\mathrm{test}},\ \mathcal{V}^\tau_{S_i} \neq \emptyset\}.$$

During training, we set $\mathcal{S}^\tau = \mathcal{S}^\tau_{\mathrm{train}}$, so the views contain only training targets. During validation, we set $\mathcal{S}^\tau = \mathcal{S}^\tau_{\mathrm{train}} \cup \mathcal{S}^\tau_{\mathrm{val}}$, retaining the training targets and adding the validation targets. During testing, we similarly set $\mathcal{S}^\tau = \mathcal{S}^\tau_{\mathrm{train}} \cup \mathcal{S}^\tau_{\mathrm{test}}$, retaining the training targets and adding the test targets. Thus, validation and test targets enter the views only at their respective evaluation stages. At either stage, training targets participate in message passing alongside the targets being evaluated, but neither set's labels are supplied as input. Predictions are evaluated only for the validation or test targets, respectively. The updates at each layer produce type-specific representations for the targets in these views.

C.12 Computation of Subgraph Representations in Minibatches

All six selected CTAP configurations use Jumping Knowledge concatenation across layers. The expressions below define the final-layer representation $\mathbf{h}_{S_i}$; classification uses the layer-concatenated representation $\mathbf{z}_{S_i}$ defined in Appendix C.13, in training, validation, and testing.

Given a training minibatch $\mathcal{S}_B \subseteq \mathcal{S}_{\text{train}}$, the views retain all targets in $\mathcal{S}^{\tau}_{\text{train}}$. For each subgraph in this minibatch, we combine its type-specific representations as in Section 5.4:

$$\mathbf{h}_{S_i} = \text{SF}_{\text{subgraph}}(\mathbf{H}^{(L)}_{S_i}), \qquad S_i \in \mathcal{S}_B.$$
$$\hat{\mathbf{y}}_{S_i} = f_{\text{cls}}(\mathbf{z}_{S_i}), \qquad S_i \in \mathcal{S}_B.$$

The minibatch loss is

$$\mathcal{L}_B = \frac{1}{|\mathcal{S}_B|} \sum_{S_i \in \mathcal{S}_B} \ell_{\text{cls}}(\hat{\mathbf{y}}_{S_i}, \mathbf{y}_{S_i}).$$

During validation, fusion across node types and classification are applied to the validation targets together:

$$\mathbf{h}_{S_i} = \text{SF}_{\text{subgraph}}(\mathbf{H}^{(L)}_{S_i}), \qquad S_i \in \mathcal{S}_{\text{val}}.$$
$$\hat{\mathbf{y}}_{S_i} = f_{\text{cls}}(\mathbf{z}_{S_i}), \qquad S_i \in \mathcal{S}_{\text{val}}.$$

During testing, the corresponding targets are

$$\mathbf{h}_{S_i} = \text{SF}_{\text{subgraph}}(\mathbf{H}^{(L)}_{S_i}), \qquad S_i \in \mathcal{S}_{\text{test}}.$$
$$\hat{\mathbf{y}}_{S_i} = f_{\text{cls}}(\mathbf{z}_{S_i}), \qquad S_i \in \mathcal{S}_{\text{test}}.$$

The validation and test labels are used for evaluation, not for computing $\mathcal{L}_B$.

C.13 Application of Jumping Knowledge

We apply the concatenation variant of Jumping Knowledge (Xu et al., 2018) to layer-wise subgraph representations. For each layer $\ell = 1, \ldots, L$, we fuse the type-specific target representations using the input-matrix construction in Appendix C.8:

$$\mathbf{h}^{(\ell)}_{S_i} = \text{SF}^{(\ell)}_{\text{subgraph}}(\mathbf{H}^{(\ell)}_{S_i}).$$

The fusion module is separately parameterized at each layer. We then concatenate these fused representations along the feature dimension:

$$\mathbf{z}_{S_i} = [\mathbf{h}^{(1)}_{S_i}; \ldots; \mathbf{h}^{(L)}_{S_i}], \qquad \hat{\mathbf{y}}_{S_i} = f_{\text{cls}}(\mathbf{z}_{S_i}).$$

Thus, fusion across node types precedes concatenation across layers; initial input features are not included. Whether to use this concatenation is a tuned hyperparameter. Without it, the classifier receives only $\mathbf{h}^{(L)}_{S_i}$, denoted by $\mathbf{h}_{S_i}$ in Section 5.4.

C.14 Subgraph Classification

We evaluate subgraph representation learning methods using the subgraph classification task. Given class labels $\mathcal{Y} = \{\mathbf{y}_{S_1}, \mathbf{y}_{S_2}, \ldots, \mathbf{y}_{S_n}\}$ for the subgraphs in $\mathcal{S}$, each $\mathbf{y}_{S_i} \in \{0,1\}^{d_y}$ is a binary vector, where d_y is the number of classes. Each entry is one if the corresponding class applies and zero otherwise. It is one-hot for single-label classification and multi-hot for multi-label classification. We compute the logits from the learned representation as $\hat{\mathbf{y}}_{S_i} = f_{\text{cls}}(\mathbf{h}_{S_i})$, where f_{cls} is a classification head and $\hat{\mathbf{y}}_{S_i}$ is the logits vector for subgraph S_i, with one unnormalized score per class. Its ground-truth label vector $\mathbf{y}_{S_i}$ has the same dimension as $\hat{\mathbf{y}}_{S_i}$.

The classification loss $\ell_{\text{cls}}(\cdot,\cdot)$ compares the logits with $\mathbf{y}_{S_i}$ using softmax cross-entropy for single-label tasks or sigmoid binary cross-entropy averaged over classes for multi-label tasks. We jointly train CTAP and the classification head end-to-end by minimizing

$$\mathcal{L} = \frac{1}{|\mathcal{S}_{\text{train}}|} \sum_{S_i \in \mathcal{S}_{\text{train}}} \ell_{\text{cls}}(\hat{\mathbf{y}}_{S_i}, \mathbf{y}_{S_i}), \tag{41}$$

where $\mathcal{S}_{\text{train}}$ is the set of training subgraphs.

C.15 SUBGRAPH-TO-NODE (S2N) TRANSLATION

Given a base graph G and a collection of subgraphs $\mathcal{S}$, Subgraph-To-Node (S2N) translation (Kim & Oh, 2024) maps each subgraph S_i to a node $\hat{v}_i = T_v(S_i)$ and the relation between each pair of subgraphs S_i, S_j to an edge weight $T_e(S_i, S_j)$. The translated graph is $\widehat{G} = (\widehat{\mathcal{V}}, \widehat{\mathbf{A}})$, with $\widehat{\mathcal{V}} = \{T_v(S_i) \mid S_i \in \mathcal{S}\}$ and $\widehat{\mathbf{A}}_{ij} = T_e(S_i, S_j)$, where $T_v(\cdot)$ and $T_e(\cdot, \cdot)$ are the functions for translating nodes and edges, respectively. Each of the n subgraphs corresponds to one node in $\widehat{\mathcal{V}}$, and $\widehat{\mathbf{A}} \in \mathbb{R}^{n \times n}$ is the weighted adjacency matrix under the ordering $S_1, \ldots, S_n$. The representation learned for $\hat{v}_i$ in $\widehat{G}$ serves as the representation of subgraph S_i.

D COMPUTATIONAL COMPLEXITY OF CTAP

CTAP incurs costs for constructing G^τ and S_i^τ, constructing the view matrices, and updating node and subgraph representations. We derive each cost below and then combine them to obtain the overall complexity. The bounds count arithmetic operations; the costs of message passing depend on the chosen MPNN.

D.1 TIME COMPLEXITY OF CONSTRUCTING G^τ AND S_i^τ

The steps for constructing G^τ and the corresponding S_i^τ across node types are as follows.

First, we assign each base-graph node to $\mathcal{V}^\tau$ according to its type and assign its local index. This costs $O(|\mathcal{V}|)$ across all types. We then scan $\mathcal{E}$ and place each same-type edge in the corresponding $\mathcal{E}^\tau$, which costs $O(|\mathcal{E}|)$. These steps construct the node and edge sets of all G^τ in $O(|\mathcal{V}| + |\mathcal{E}|)$ time.

Next, we partition each target's node set $\mathcal{V}_{S_i}$ by type to obtain its nonempty $\mathcal{V}_{S_i}^\tau$. This costs $O(\sum_{S_i \in \mathcal{S}} |\mathcal{V}_{S_i}|)$ across all targets. If the input target graphs explicitly provide $\mathcal{E}_{S_i}$, we likewise partition those edges to obtain $\mathcal{E}_{S_i}^\tau$ at a cost of $O(\sum_{S_i \in \mathcal{S}} |\mathcal{E}_{S_i}|)$. Thus, constructing the node and edge sets of all S_i^τ from explicitly supplied target graphs takes

$$O\left(\sum_{S_i \in \mathcal{S}} (|\mathcal{V}_{S_i}| + |\mathcal{E}_{S_i}|)\right).$$

D.2 TIME COMPLEXITY OF CONSTRUCTING THE VIEWS

Constructing the views as defined in the main text requires their membership, adjacency, and initial feature matrices. The costs of these computations are as follows. The translated adjacency matrices would be computed once for each fixed target set and reused across layers. Our implementation in Appendix C.6 instead evaluates their products with features using the membership matrices, so it does not incur the costs of constructing the translated adjacencies derived below.

Time Complexity of Constructing $\mathbf{M}^{\tau,r}$ The steps for constructing all membership matrices $\mathbf{M}^{\tau,r}$ from the type-specific target and projected node sets are as follows. First, we can construct all $\mathbf{M}_S^{\tau,r}$ by scanning the target node sets $\mathcal{V}_{S_i}$ once, using each node's type and prepared local index. Thus, constructing all $\mathbf{M}_S^{\tau,r}$ costs $O(\sum_{S_i \in \mathcal{S}} |\mathcal{V}_{S_i}|)$. Then, we can construct all $\mathbf{M}_P^{\tau,r}$ by scanning $\mathcal{E}$ once and assigning each cross-type edge to the appropriate entries according to its endpoint types and relation. Thus, constructing all $\mathbf{M}_P^{\tau,r}$ costs $O(|\mathcal{E}|)$.

Combining these two costs, constructing all membership matrices $\mathbf{M}^{\tau,r}$ takes

$$O\left(\sum_{S_i \in \mathcal{S}} |\mathcal{V}_{S_i}| + |\mathcal{E}|\right).$$

Time Complexity of Computing $\widehat{\mathbf{A}}_{\text{node}}^{\tau,r}$ To compute $\widehat{\mathbf{A}}_{\text{node}}^{\tau,r}$, we evaluate $\mathbf{M}^{\tau,r}(\mathbf{M}^{\tau,r})^\top$ by accumulating $\sum_{v \in \mathcal{V}^\tau} (c_v^{\tau,r})^2$ pair contributions, where $c_v^{\tau,r}$ is the number of nonzero entries in the column for v. Each such column contributes to $(c_v^{\tau,r})^2$ adjacency entries, counting repeated contributions to the same entry. We accumulate these contributions with updates to a hash table that take constant time in expectation, so this step costs $O(\sum_{v \in \mathcal{V}^\tau} (c_v^{\tau,r})^2)$.

Time Complexity of Computing $\widehat{\mathbf{A}}^{\tau,r}_{\text{edge}}$ To compute $\widehat{\mathbf{A}}^{\tau,r}_{\text{edge}}$, we evaluate $\mathbf{M}^{\tau,r}\mathbf{A}^{\tau}(\mathbf{M}^{\tau,r})^{\top}$ by accumulating $c^{\tau,r}_u c^{\tau,r}_v$ pair contributions for each nonzero entry (u,v) of $\mathbf{A}^{\tau}$. With the same accumulation assumption, their total cost is $O(\sum_{(u,v):\mathbf{A}^{\tau}_{uv}\neq 0} c^{\tau,r}_u c^{\tau,r}_v)$. The same-type view directly reuses G^{τ} and does not require enumerating pairs of translated nodes.

Time Complexity of Constructing $\widehat{\mathbf{X}}^{\tau,r}$ To construct $\widehat{\mathbf{X}}^{\tau,r}$ with input feature width d_0, we sum the features of each target subgraph's member nodes. This costs $O(\sum_{S^{\tau}_i\in\mathcal{S}^{\tau}}|\mathcal{V}^{\tau}_{S_i}|d_0)$. The projected subgraphs use their anchor features directly. The target features can be shared across relations for the same type.

D.3 Time Complexity of an MPNN Layer

We first derive the costs of $\mathrm{Aggregate}(\cdot,\cdot)$ using the adjacency matrices of the views, then derive the costs of evaluating the same products in our implementation. The input features have width d. The number of nonzero entries in $\mathbf{A}^{\tau}$ is $2|\mathcal{E}^{\tau}|$ minus the number of self-loops in $\mathcal{E}^{\tau}$. The number of nonzero entries in $\widehat{\mathbf{A}}^{\tau,r}_{\text{node}}$ is $2|\widehat{\mathcal{E}}^{\tau,r}_{\text{node}}|$ minus the number of self-loops in $\widehat{\mathcal{E}}^{\tau,r}_{\text{node}}$. The number of nonzero entries in $\widehat{\mathbf{A}}^{\tau,r}_{\text{edge}}$ is $2|\widehat{\mathcal{E}}^{\tau,r}_{\text{edge}}|$ minus the number of self-loops in $\widehat{\mathcal{E}}^{\tau,r}_{\text{edge}}$. Each edge set here includes its self-loops. An undirected non-self edge contributes two entries, whereas a self-loop contributes one; thus each count has the same asymptotic order as its corresponding edge count.

For $\widehat{\mathbf{A}}^{\tau,r}_{\text{node}}\mathbf{H}^{\tau,r,(\ell-1)}_{\text{node}}$, each nonzero adjacency entry weights a feature vector and adds it to the corresponding output row. This requires $O(d)$ operations per nonzero entry, giving $O(|\widehat{\mathcal{E}}^{\tau,r}_{\text{node}}|d)$. For $\widehat{\mathbf{A}}^{\tau,r}_{\text{edge}}\mathbf{H}^{\tau,r,(\ell-1)}_{\text{edge}}$, the same calculation requires $O(d)$ operations for each nonzero entry of $\widehat{\mathbf{A}}^{\tau,r}_{\text{edge}}$, giving $O(|\widehat{\mathcal{E}}^{\tau,r}_{\text{edge}}|d)$. Our implementation evaluates the right-hand sides in Appendix C.6. The number of nonzero entries in $\mathbf{M}^{\tau,r}$ is $\sum_{S^{\tau}_i\in\mathcal{S}^{\tau}}|\mathcal{V}^{\tau}_{S_i}|+\sum_{q\in\mathcal{Q}^{\tau,r}}|\mathcal{V}^{\tau,r}_{q}|$. Multiplying first by $(\mathbf{M}^{\tau,r})^{\top}$ and then by $\mathbf{M}^{\tau,r}$ each requires d operations per membership entry, so the node-overlap product costs $O((\sum_{S^{\tau}_i\in\mathcal{S}^{\tau}}|\mathcal{V}^{\tau}_{S_i}|+\sum_{q\in\mathcal{Q}^{\tau,r}}|\mathcal{V}^{\tau,r}_{q}|)d)$. For the edge-connectivity product, the intervening multiplication by $\mathbf{A}^{\tau}$ adds $O(|\mathcal{E}^{\tau}|d)$, giving $O((\sum_{S^{\tau}_i\in\mathcal{S}^{\tau}}|\mathcal{V}^{\tau}_{S_i}|+\sum_{q\in\mathcal{Q}^{\tau,r}}|\mathcal{V}^{\tau,r}_{q}|+|\mathcal{E}^{\tau}|)d)$. $\mathrm{Update}(\cdot)$ is evaluated once for each of the $|\widehat{\mathcal{V}}^{\tau,r}|$ nodes in a translated view. A d-dimensional linear layer requires $O(d^2)$ operations per node. A fixed-depth MLP with d-dimensional layers has the same order of cost, while its pointwise activations cost $O(d)$ per node. The cost of updating the nodes in the view is therefore $O(|\widehat{\mathcal{V}}^{\tau,r}|d^2)$. For the same-type view, $\mathrm{Aggregate}(\cdot,\cdot)$ costs $O(|\mathcal{E}^{\tau}|d)$ and $\mathrm{Update}(\cdot)$ costs $O(|\mathcal{V}^{\tau}|d^2)$.

For updates, substitute $|\widehat{\mathcal{V}}^{\tau,r}| = |\mathcal{S}^{\tau}| + |\mathcal{Q}^{\tau,r}|$. Summing the target-node counts over relations gives $|\mathcal{C}^{\tau}||\mathcal{S}^{\tau}|$ for each type. Summing over projected nodes counts each original node q once per incident cross-type relation, giving $\sum_{q\in\mathcal{V}}|\mathcal{R}_q|$, with $\mathcal{R}_q$ defined in Appendix C.8. The same-type views contain at most $|\mathcal{V}|$ nodes in total. The node-overlap and edge-connectivity views duplicate the terms for updating translated nodes at most twice, which does not change their Big-O order. The cost of an MPNN for each view is the sum of its $\mathrm{Aggregate}(\cdot,\cdot)$ and $\mathrm{Update}(\cdot)$ costs. Evaluating the left-hand sides with the translated adjacency matrices, and including the same-type views, gives the following MPNN cost per CTAP layer:

$$O\left(d\sum_{\tau}\left[|\mathcal{E}^{\tau}|+\sum_{r}\left(|\widehat{\mathcal{E}}^{\tau,r}_{\text{node}}|+|\widehat{\mathcal{E}}^{\tau,r}_{\text{edge}}|\right)\right]+d^2\left[|\mathcal{V}|+\sum_{q}|\mathcal{R}_q|+\sum_{\tau}|\mathcal{C}^{\tau}||\mathcal{S}^{\tau}|\right]\right),$$

where τ ranges over $\mathcal{T}_{\mathcal{V}}$, r over $\mathcal{C}^{\tau}$, and q over $\mathcal{V}$. The edge-connectivity term is omitted when $\mathcal{E}^{\tau}=\emptyset$.

In our implementation, both translated views use the right-hand sides. Target memberships are counted $|\mathcal{C}^{\tau}|$ times for each type. Summing projected memberships over types and relations counts each cross-type base edge twice; together with the same-type views, this gives an $O(|\mathcal{E}|d)$ term. The multiplication by $\mathbf{A}^{\tau}$ in each edge-connectivity view contributes $|\mathcal{C}^{\tau}||\mathcal{E}^{\tau}|d$ for that type. Combining these terms with the same costs of updating nodes gives the MPNN cost T_{MPNN} per CTAP layer in our implementation:

$$O\left(d\left[|\mathcal{E}|+\sum_{\tau}|\mathcal{C}^{\tau}|\left(\sum_{S^{\tau}_i\in\mathcal{S}^{\tau}}|\mathcal{V}^{\tau}_{S_i}|+|\mathcal{E}^{\tau}|\right)\right]+d^2\left[|\mathcal{V}|+\sum_{q}|\mathcal{R}_q|+\sum_{\tau}|\mathcal{C}^{\tau}||\mathcal{S}^{\tau}|\right]\right).$$

The terms for the translated edges are thereby replaced by terms for target memberships and base edges. The costs of updating nodes are unchanged.

D.4 Time Complexity of Applying Semantic Fusion

To compute $\mathrm{SF}(\mathcal{H})$ for vectors of width d, we first compute the projections for queries, keys, and values. With query/key width $O(d)$, this costs $O(|\mathcal{H}|d^2)$. We then compute the attention scores and weighted values, which costs $O(|\mathcal{H}|^2 d)$. Finally, we apply the projection that produces the output at a cost of $O(|\mathcal{H}|d^2)$. Combining these costs, one SF call takes

$$O(|\mathcal{H}|d^2 + |\mathcal{H}|^2 d).$$

For the padded inputs to semantic fusion in Appendix C.10, $|\mathcal{H}|$ includes the masked source positions.

D.5 Time Complexity of Fusing the Outputs of the Views

We sum the cost in Appendix D.4 over the inputs defined in Section 5.2 to obtain the total cost of fusing the outputs of the views in one CTAP layer. For node q, each relation in $\mathcal{R}_q$ contributes one vector from the node-overlap view and at most one from the edge-connectivity view. The same-type view contributes at most one additional vector, so $|\mathcal{H}_q^{(\ell)}| \le 2|\mathcal{R}_q| + 1$. Substituting this count into the cost of one SF call and summing over $q \in \mathcal{V}$ gives

$$O\left(d \sum_{q \in \mathcal{V}} (1 + |\mathcal{R}_q|)(d + 1 + |\mathcal{R}_q|)\right).$$

For a target $S_i^\tau \in \mathcal{S}^\tau$, each relation in $\mathcal{C}^\tau$ contributes one node-overlap vector and at most one edge-connectivity vector, so $|\mathcal{H}_{S_i}^{\tau,(\ell)}| \le 2|\mathcal{C}^\tau|$. All $|\mathcal{S}^\tau|$ targets of the same type use the same number of views. Their combined cost, summed over types, is

$$O\left(d \sum_{\tau \in \mathcal{T}_\mathcal{V}} |\mathcal{S}^\tau||\mathcal{C}^\tau|(d + |\mathcal{C}^\tau|)\right).$$

Adding the costs for nodes and type-specific targets gives the total per layer:

$$O\left(d \left[\sum_q (1 + |\mathcal{R}_q|)(d + 1 + |\mathcal{R}_q|) + \sum_\tau |\mathcal{S}^\tau||\mathcal{C}^\tau|(d + |\mathcal{C}^\tau|)\right]\right),$$

where q ranges over $\mathcal{V}$ and τ over $\mathcal{T}_\mathcal{V}$. This counts the fusion in Section 5.2; the fusion within membership pooling is included in Appendix D.6.

D.6 Time Complexity of Performing Membership Pooling

To perform membership pooling, we pool cross-type neighbor features for each node and constituent-node features for each target subgraph, then fuse each pooled output with its corresponding view-fused representation. For node q, we first pool its neighbor features separately for each $r \in \mathcal{R}_q$. Using mean, mean, or size pooling over $\mathcal{V}_q^{\tau,r}$ costs $O(|\mathcal{V}_q^{\tau,r}|d)$ per relation. We then fuse these pooled vectors with the view-fused representation of q. This SF call has $1 + |\mathcal{R}_q|$ inputs and takes

$$O\big((1 + |\mathcal{R}_q|)d^2 + (1 + |\mathcal{R}_q|)^2 d\big).$$

For $S_i^\tau \in \mathcal{S}^\tau$, we first pool its constituent-node features at a cost of $O(|\mathcal{V}_{S_i}^\tau|d)$. We then fuse that pooled vector with the corresponding view-fused representation. This SF call has two inputs and costs $O(d^2)$. We obtain the total cost by adding the costs of these pooling and fusion operations over all relevant nodes, relations, and type-specific target subgraphs.

D.7 Total Time Complexity of One CTAP Layer

One CTAP layer performs message passing on the views, fuses their outputs, and performs membership pooling, including its subsequent fusion. The decomposition and construction costs derived above prepare the graphs, memberships, and initial features for a fixed target set; they are not repeated at each layer. We combine the costs of the three repeated operations below for our implementation using membership matrices, with feature dimension d.

For membership pooling in Appendix D.6, summing the cross-type neighbor counts over nodes and relations counts each cross-type base edge twice. Summing the constituent-node counts over types counts each target membership once. Thus, pooling without the subsequent fusion takes

$$O\left(d\left[|\mathcal{E}| + \sum_{S_i \in \mathcal{S}} |\mathcal{V}_{S_i}|\right]\right).$$

For nodes, the fusion after pooling has $1 + |\mathcal{R}_q|$ inputs, and its cost has the same order as the fusion of the outputs of the views in Appendix D.5. For each type-specific target, the two-input fusion after pooling adds $O(d^2)$ to the cost of fusing its views. Adding these fusion costs gives

$$O\left(d\left[\sum_q (1+|\mathcal{R}_q|)(d+1+|\mathcal{R}_q|) + \sum_\tau |\mathcal{S}^\tau|(1+|\mathcal{C}^\tau|)(d+1+|\mathcal{C}^\tau|)\right]\right).$$

Here q ranges over $\mathcal{V}$ and τ over $\mathcal{T}_\mathcal{V}$. The cost of one SF call derived in Appendix D.4 is already included in these totals, rather than added as a separate operation.

When the same pair of distinct node types is not connected by multiple relation types, $|\mathcal{R}_q|, |\mathcal{C}^\tau| \leq |\mathcal{T}_\mathcal{V}| - 1$: each node type can be connected to at most $|\mathcal{T}_\mathcal{V}| - 1$ other types, with at most one relation type for each. The following bound applies under this condition, which holds for the six datasets used here. Moreover, $\sum_\tau |\mathcal{S}^\tau| \leq |\mathcal{T}_\mathcal{V}||\mathcal{S}|$ and $\sum_\tau \sum_{S_i^\tau \in \mathcal{S}^\tau} |\mathcal{V}_{S_i}^\tau| = \sum_{S_i \in \mathcal{S}} |\mathcal{V}_{S_i}|$. Substituting these counts into the MPNN cost in Appendix D.3 bounds its products by $O(|\mathcal{T}_\mathcal{V}|d(|\mathcal{E}| + \sum_{S_i \in \mathcal{S}} |\mathcal{V}_{S_i}|))$ and its updates by $O(|\mathcal{T}_\mathcal{V}|d^2(|\mathcal{V}| + |\mathcal{T}_\mathcal{V}||\mathcal{S}|))$. The combined fusion cost above is bounded by $O(|\mathcal{T}_\mathcal{V}|d(d + |\mathcal{T}_\mathcal{V}|)(|\mathcal{V}| + |\mathcal{T}_\mathcal{V}||\mathcal{S}|))$; this also covers padding to the available views and relations. Adding the MPNN, pooling, and fusion costs and collecting their shared terms yields the total time complexity of one CTAP layer:

$$O\left(|\mathcal{T}_\mathcal{V}|d\left[|\mathcal{E}| + \sum_{S_i \in \mathcal{S}} |\mathcal{V}_{S_i}| + (|\mathcal{V}| + |\mathcal{T}_\mathcal{V}||\mathcal{S}|)(d + |\mathcal{T}_\mathcal{V}|)\right]\right).$$

For a fixed number of types and relations, this simplifies to

$$O\left(d\left[|\mathcal{E}| + \sum_{S_i \in \mathcal{S}} |\mathcal{V}_{S_i}|\right] + d^2(|\mathcal{V}| + |\mathcal{S}|)\right).$$

The first term accounts for processing base edges and target memberships; the second accounts for updating representations and fusing features. Thus, at fixed feature dimension and type and relation counts, the cost of a layer is linear in the base graph size and the total size of the supplied target member sets. It does not depend on enumerating all translated edges.

E Total Size of the Views

The cost of an MPNN layer in Appendix D.3 depends on the total numbers of nodes and edges in the views. We count these nodes exactly and bound the number of edges with explicit coefficients, then measure both quantities on the six datasets. For a view G', $\mathcal{V}_{G'}$ and $\mathcal{E}_{G'}$ denote its node and edge sets. We count an undirected edge once, including a self-loop once, as in Appendix D.3.

E.1 Total Number of Nodes

Each translated view contains $|\mathcal{S}^\tau|$ target nodes and $|\mathcal{Q}^{\tau,r}|$ projected nodes. The node-overlap view is always present; the edge-connectivity view has the same nodes and is present when $\mathcal{E}^\tau \neq \emptyset$. The same-type views partition the nodes whose types have internal edges. Thus, the exact total is

$$\sum_{G' \in \mathcal{G}_{\text{view}}} |\mathcal{V}_{G'}| = \sum_{\tau : \mathcal{E}^\tau \neq \emptyset} |\mathcal{V}^\tau| + \sum_{\tau,r} \big(1 + \mathbf{1}[\mathcal{E}^\tau \neq \emptyset]\big)(|\mathcal{S}^\tau| + |\mathcal{Q}^{\tau,r}|).$$

Here and below, τ ranges over $\mathcal{T}_{\mathcal{V}}$ and r over $\mathcal{C}^{\tau}$; $\mathbf{1}[\cdot]$ equals one when its condition holds and zero otherwise. Each node q appears as an anchor once for each relation in $\mathcal{R}_q$, giving $\sum_{\tau,r} |\mathcal{Q}^{\tau,r}| = \sum_{q\in\mathcal{V}} |\mathcal{R}_q|$. The target nodes of type τ appear once per incident relation. Replacing each multiplicity by at most two gives

$$\sum_{G'\in\mathcal{G}_{\text{view}}} |\mathcal{V}_{G'}| \le |\mathcal{V}| + 2\sum_{q\in\mathcal{V}} |\mathcal{R}_q| + 2\sum_{\tau} |\mathcal{C}^{\tau}||\mathcal{S}^{\tau}|.$$

This bound retains the actual numbers of incident relations and type-specific targets rather than multiplying every view by the size of the entire graph.

In the six datasets, a cross-type relation is identified by its pair of endpoint types. Consequently, both $|\mathcal{R}_q|$ and $|\mathcal{C}^{\tau}|$ are at most $|\mathcal{T}_{\mathcal{V}}| - 1$. Each target has at most one nonempty part per type and at most one nonempty part per member node. Therefore, $\sum_{\tau} |\mathcal{S}^{\tau}| \le \min\{|\mathcal{T}_{\mathcal{V}}||\mathcal{S}|, \sum_{S_i\in\mathcal{S}} |\mathcal{V}_{S_i}|\}$, which yields

$$\sum_{G'\in\mathcal{G}_{\text{view}}} |\mathcal{V}_{G'}| \le (2|\mathcal{T}_{\mathcal{V}}| - 1)|\mathcal{V}| + 2(|\mathcal{T}_{\mathcal{V}}| - 1)\min\left\{|\mathcal{T}_{\mathcal{V}}||\mathcal{S}|, \sum_{S_i\in\mathcal{S}} |\mathcal{V}_{S_i}|\right\}.$$

In particular, relative to $|\mathcal{V}| + \sum_{S_i\in\mathcal{S}} |\mathcal{V}_{S_i}|$, the total number of nodes is at most $2|\mathcal{T}_{\mathcal{V}}| - 1$ times this input count. This coefficient is three, five, and thirteen for two, three, and seven types, respectively. If several relations connect the same pair of types, the preceding bound using $\mathcal{R}_q$ and $\mathcal{C}^{\tau}$ still applies.

E.2 Upper Bounds on the Total Number of Edges

We use $c_v^{\tau,r}$ from Appendix D.2, the number of target and projected subgraphs containing v. For the node-overlap view, a member node v can connect at most $\binom{c_v^{\tau,r}}{2}$ distinct pairs of translated nodes. Every translated node has a self-loop because its membership set is nonempty. Counting repeated contributions to an edge only once can only decrease the total, so

$$|\widehat{\mathcal{E}}^{\tau,r}_{\text{node}}| \le \min\left\{\binom{|\widehat{\mathcal{V}}^{\tau,r}| + 1}{2}, |\widehat{\mathcal{V}}^{\tau,r}| + \sum_{v\in\mathcal{V}^{\tau}} \binom{c_v^{\tau,r}}{2}\right\}.$$

For the edge-connectivity view, a non-self edge $\{u, v\}$ of the same-type graph contributes at most $c_u^{\tau,r} c_v^{\tau,r}$ translated edges, including possible self-loops. The base graphs in our datasets have no self-loops. Thus,

$$|\widehat{\mathcal{E}}^{\tau,r}_{\text{edge}}| \le \min\left\{\binom{|\widehat{\mathcal{V}}^{\tau,r}| + 1}{2}, \sum_{\{u,v\}\in\mathcal{E}^{\tau}} c_u^{\tau,r} c_v^{\tau,r}\right\}.$$

If a base graph has a self-loop at v, its contribution is bounded by $\binom{c_v^{\tau,r}+1}{2}$ instead. To bound the total over all views, we add the node-overlap bound for every (τ, r), the edge-connectivity bound for those with $\mathcal{E}^{\tau} \neq \emptyset$, and $\sum_{\tau} |\mathcal{E}^{\tau}|$ for the same-type views. These bounds require only membership counts and the base edges, without enumerating the translated edges. Without the complete-graph caps, their sum can be written as

$$\sum_{G'} |\mathcal{E}_{G'}| \le \sum_{\tau} |\mathcal{E}^{\tau}| + \sum_{\tau,r}\left[|\widehat{\mathcal{V}}^{\tau,r}| + \sum_{v} \binom{c_v^{\tau,r}}{2} + \sum_{\{u,v\}\in\mathcal{E}^{\tau}} c_u^{\tau,r} c_v^{\tau,r}\right].$$

Here G' ranges over $\mathcal{G}_{\text{view}}$ and v over $\mathcal{V}^{\tau}$. The last sum is empty when the corresponding edge-connectivity view is absent. The measured bounds below also apply the complete-graph cap separately to each translated view.

Unlike the bound on nodes, these bounds depend on how frequently base nodes occur in the subgraphs and on which such nodes are connected. Type counts alone do not bound this increase by a constant: if n distinct two-node targets share one member node, their total membership count is $2n$, but their translated nodes form $n(n-1)/2$ non-self edges in the node-overlap view.

E.3 Measured Sizes on the Six Datasets

We construct the views defined in Section 4.2 using the target sets for training, validation, and testing in Appendix C.11. The counts describe the view matrices before the unit self-loops used by

Table 5: Sizes of all views for each target set specified by Split, with $G' \in \mathcal{G}_{\text{view}}$ in every sum over G'. Occurrences in different views are counted separately; an undirected edge or self-loop is counted once per view. The upper bounds are evaluated as specified in Appendix E.3. Here, train, train+val, and train+tst denote $\mathcal{S} = \mathcal{S}_{\text{train}}$, $\mathcal{S} = \mathcal{S}_{\text{train}} \cup \mathcal{S}_{\text{val}}$, and $\mathcal{S} = \mathcal{S}_{\text{train}} \cup \mathcal{S}_{\text{test}}$, respectively; these are the target sets used to construct the views and compute the final ratio.

Dataset	Split	$\sum_{G'} \lvert\mathcal{V}_{G'}\rvert$	Upper bound on $\sum_{G'} \lvert\mathcal{V}_{G'}\rvert$	$\sum_{G'} \lvert\mathcal{E}_{G'}\rvert$	Upper bound on $\sum_{G'} \lvert\mathcal{E}_{G'}\rvert$	$\frac{\sum_{G'} \lvert\mathcal{E}_{G'}\rvert}{\lvert\mathcal{E}\rvert}$	$\frac{\sum_{G'} (\lvert\mathcal{V}_{G'}\rvert + \lvert\mathcal{E}_{G'}\rvert)}{\lvert\mathcal{V}\rvert + \lvert\mathcal{E}\rvert + \sum_{S_i \in \mathcal{S}} \lvert\mathcal{V}_{S_i}\rvert}$
DBLP	train	86,474	210,420	97,823,452	109,633,202	364.51×	247.33×
	train+val	88,442	214,356	103,680,374	116,271,137	386.34×	255.10×
	train+tst	88,506	214,484	103,943,773	116,584,659	387.32×	255.46×
Movie	train	220,211	624,633	22,779,307	25,179,851	48.60×	25.14×
	train+val	222,211	628,633	24,215,026	26,812,976	51.67×	25.81×
	train+tst	222,211	628,633	24,237,127	26,832,924	51.71×	25.78×
Rare Disease Category	train	574,136	574,136	584,319,576	1,384,420,453	360.61×	309.31×
	train+val	597,320	597,320	617,964,704	1,493,874,821	381.37×	323.59×
	train+tst	597,320	597,320	618,689,641	1,494,663,202	381.82×	323.97×
Rare Disease Inheritance	train	478,472	478,472	479,521,888	1,014,124,954	295.93×	263.92×
	train+val	489,688	489,688	495,127,618	1,056,910,025	305.57×	270.77×
	train+tst	489,688	489,688	495,531,817	1,057,387,700	305.82×	270.94×
Protein Set Function	train	33,160	33,160	10,644,811	24,392,911	206.37×	80.53×
	train+val	33,562	33,562	10,830,854	25,294,854	209.97×	76.44×
	train+tst	33,572	33,572	10,821,145	25,264,088	209.79×	77.33×
Gene Set Process	train	51,391	80,773	8,842,659	13,199,110	162.95×	52.09×
	train+val	51,919	81,829	8,944,138	13,349,902	164.82×	49.84×
	train+tst	51,928	81,847	8,946,441	13,351,785	164.87×	49.68×

the encoders are applied. We deduplicate undirected base edges and count each nonzero translated edge once, regardless of its weight or the number of contributions that produce it.

In Table 5, *Split* specifies the target set used to construct the views, as defined in its caption. The train, train+val, and train+tst rows correspond to Training, Validation, and Testing in Table 6, respectively. The base graph, projected subgraphs, and number of views are the same in all three stages. Each row describes one construction of these views, not a sum over minibatches, epochs, or repeated runs.

In Table 5, $\sum_{G'} |\mathcal{V}_{G'}|$ adds the nodes in every same-type and translated view, where $G' \in \mathcal{G}_{\text{view}}$. The upper bound on $\sum_{G'} |\mathcal{V}_{G'}|$ substitutes the row's target set into $|\mathcal{V}| + 2\sum_{q \in \mathcal{V}} |\mathcal{R}_q| + 2\sum_{\tau} |\mathcal{C}^{\tau}||\mathcal{S}^{\tau}|$; it uses the observed incident relations rather than the looser bound based only on the number of types. The sum $\sum_{G'} |\mathcal{E}_{G'}|$ counts the edges over these views. For each translated view, we count positive entries on or above the diagonal of its defined adjacency matrix. This counts each undirected edge and each self-loop once, even if several members or base edges contribute to its weight. We add the same-type edges to these counts. A node or edge occurring in several views is counted separately in each view. The upper bound on $\sum_{G'} |\mathcal{E}_{G'}|$ adds the smaller of the complete-graph bound and the membership-based bound for each translated view, using the stage-specific $c_v^{\tau,r}$, then adds $\sum_{\tau} |\mathcal{E}^{\tau}|$ for the same-type views.

The denominator of the final ratio uses the target set of that row's stage. It counts the base nodes, distinct base edges, and target memberships supplied as input; it does not include projected memberships or separately count the induced edges of each target. Both ratios use the measured totals, not the upper bounds, and are rounded to two decimal places. They compare counts of graph elements and memberships, not memory in bytes.

Across the six datasets and three stages, the total number of view nodes is 1.20–5.68 times the number of base nodes, or 0.37–2.41 times the number of base nodes plus target memberships. The bounds on edges are 1.11–2.42 times the measured totals. However, the totals themselves are 48.60–387.32 times the number of base edges. Thus, the measured increase in nodes is limited, whereas the translated edges can substantially increase the size processed by message passing. The bounds on edges describe this increase more closely than a complete-graph bound, but do not imply that the total size is close to that of the base graph.

E.4 Sizes of the Products Using Membership Matrices

Our implementation evaluates the right-hand sides in Appendix C.6, computing the same weighted sums without constructing the translated adjacency matrices. To compare its work with multiplying the defined adjacencies by features, we count the nonzero matrix entries processed in one layer, including repeated use of a matrix. Each entry contributes to d feature coordinates. An undirected non-self edge contributes two adjacency entries, while a self-loop contributes one; thus, the adjacency counts here differ from the undirected edge counts in Table 5.

Each node-overlap view processes the entries of $\mathbf{M}^{\tau,r}$ twice, once in each multiplication, for a count of $2(\sum_{S_i^\tau \in \mathcal{S}^\tau} |\mathcal{V}_{S_i}^\tau| + \sum_{q \in \mathcal{Q}^{\tau,r}} |\mathcal{V}_q^{\tau,r}|)$. Each edge-connectivity view processes the same entries twice and the $2|\mathcal{E}^\tau|$ entries of $\mathbf{A}^\tau$ once. Each same-type view also processes $2|\mathcal{E}^\tau|$ entries. For the loop-free base graphs of our datasets, the exact total is

$$2\sum_{\tau,r}(1+\mathbf{1}[\mathcal{E}^\tau \neq \emptyset])\left(\sum_{S_i^\tau \in \mathcal{S}^\tau} |\mathcal{V}_{S_i}^\tau| + \sum_{q \in \mathcal{Q}^{\tau,r}} |\mathcal{V}_q^{\tau,r}|\right) + 2\sum_{\tau}(1+|\mathcal{C}^\tau|)|\mathcal{E}^\tau|.$$

In particular, this count includes a separate multiplication by $\mathbf{A}^\tau$ for every relation whose edge-connectivity view uses it.

Summing projected memberships over both endpoint types counts each cross-type base edge twice. The two membership multiplications and the two possible translated views therefore process at most eight entries per cross-type edge. Target memberships are processed at most $4|\mathcal{C}^\tau|$ times for each type. Because each pair of types defines at most one cross-type relation in these datasets, the total above is bounded by

$$\max\{8, 2|\mathcal{T}_\mathcal{V}|\}|\mathcal{E}| + 4(|\mathcal{T}_\mathcal{V}| - 1)\sum_{S_i \in \mathcal{S}} |\mathcal{V}_{S_i}|.$$

Unlike the number of translated edges, this bound is linear in the number of base edges and target memberships for a fixed number of types. It does not grow with the number of pairs of overlapping subgraphs.

Table 6 compares the exact counts for the products using the defined adjacencies and those in our implementation in Appendix C.6. *Using adjacencies* counts all nonzero entries in the same-type and translated adjacency matrices, once per view. It equals twice $\sum_{G' \in \mathcal{G}_{\text{view}}} |\mathcal{E}_{G'}|$ in Table 5 minus the total number of self-loops in those views, because off-diagonal edges have two symmetric entries. *Using memberships* is the exact total derived above: each translated view contributes twice its number of membership entries; each edge-connectivity view additionally contributes the nonzero entries of its same-type adjacency; and the same-type views contribute their adjacency entries once. Even if two views share a membership matrix or the same-type adjacency, its entries are counted again for each multiplication with that view's features. *Upper bound* substitutes the dataset's number of types, distinct base edges, and target memberships for that stage into $\max\{8, 2|\mathcal{T}_\mathcal{V}|\}|\mathcal{E}| + 4(|\mathcal{T}_\mathcal{V}| - 1)\sum_{S_i \in \mathcal{S}} |\mathcal{V}_{S_i}|$. *Adjacency/membership* divides *Using adjacencies* by *Using memberships*, rounded to two decimal places; for example, DBLP training gives $195{,}560{,}430/1{,}253{,}512 = 156.01$ after rounding.

Both counts are for a single MPNN layer across all views, without the common feature dimension d. Multiplying a count by d gives the number of scalar weighted contributions in these sparse products; the table is not a count of all floating-point operations in a layer. The counts precede normalization and the correction of self-loops, consistently with those identities. They exclude the dense transformations in $\mathrm{Update}(\cdot)$, which are the same for both computations, and the subsequent semantic fusion and membership pooling. Thus, the reduction measures the work in the sparse matrix products rather than an end-to-end speedup.

Across the six datasets and three stages, the products using membership matrices process 18.90–162.82 times fewer entries than those using translated adjacencies. This reduction explains why our implementation can avoid the large number of translated edges reported in Table 5 while computing the same weighted sums.

F Details of the Datasets

F.1 Dataset Statistics and Details

Table 6: Nonzero matrix entries processed in one MPNN layer across all views, excluding the common feature dimension d. Using adjacencies counts each view's adjacency entries once. Using memberships counts membership entries twice per translated view, same-type adjacency entries once per edge-connectivity view, and once per same-type view. Upper bound applies to Using memberships. Adjacency/membership divides the first count by the second; it is not a measured speedup. Appendix E.4 gives the calculation of every column. Stage uses the same target sets as Table 5.

Dataset	Stage	Using adjacencies	Using memberships	Upper bound	Adjacency/membership
DBLP	Training	195,560,430	1,253,512	2,867,112	156.01×
	Validation	207,272,306	1,275,340	2,954,424	162.52×
	Testing	207,799,040	1,276,248	2,958,056	162.82×
Movie	Training	45,338,403	2,398,700	5,845,312	18.90×
	Validation	48,207,841	2,462,462	6,100,360	19.58×
	Testing	48,252,043	2,466,666	6,117,176	19.56×
Rare Disease Category	Training	1,168,344,099	15,988,194	26,654,096	73.08×
	Validation	1,235,621,855	16,176,402	27,149,168	76.38×
	Testing	1,237,071,693	16,176,370	27,149,072	76.47×
Rare Disease Inheritance	Training	958,800,101	15,314,826	24,920,984	62.61×
	Validation	990,005,429	15,419,450	25,201,208	64.20×
	Testing	990,813,845	15,421,762	25,208,144	64.25×
Protein Set Function	Training	21,271,681	604,352	690,332	35.20×
	Validation	21,643,384	642,484	728,464	33.69×
	Testing	21,623,957	635,408	721,388	34.03×
Gene Set Process	Training	17,662,177	482,378	1,061,272	36.61×
	Validation	17,864,607	501,866	1,139,224	35.60×
	Testing	17,869,204	503,182	1,144,488	35.51×

Table 7: Statistics of the six datasets. $|\mathcal{E}|$ counts distinct undirected edges after deduplication.

Dataset	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{E}\rvert$	$\lvert\mathcal{S}\rvert$	d_y
DBLP	37,472	268,367	20,000	21
Movie	184,231	468,686	20,000	20
Rare Disease Category	105,220	1,620,364	14,482	13
Rare Disease Inheritance	105,220	1,620,364	7,007	7
Protein Set Function	11,602	51,582	2,034	18
Gene Set Process	38,081	54,265	4,717	9

Table 7 summarizes the statistics of the six datasets. Each dataset is divided into training, validation, and test sets using an approximately 8:1:1 ratio.

DBLP The DBLP dataset represents each paper through either its author set or its term set and predicts the publication venue. It contains 20,000 samples, equally divided into 10,000 author-view and 10,000 term-view subgraphs. Venue nodes are excluded from the input graph to prevent label leakage.

Movie The Movie dataset represents each movie through either its actors or its descriptive keywords and predicts its genres. It contains 20,000 samples, equally divided between the two views. Since a movie may belong to multiple genres, this is a 20-class multi-label task. In total, 14,063 samples have more than one label. Genre nodes are never included in input subgraphs, and genre edges are omitted from the input graph to prevent label leakage.

Rare Disease Category Rare Disease Category represents a patient as a node-induced subgraph containing the patient's observed positive phenotypes and known causal gene. The base graph is a heterogeneous biomedical knowledge graph with seven node types: genes/proteins, phenotypes, diseases, biological processes, molecular functions, cellular components, and pathways. The task predicts one of 13 broad disease categories from 14,482 patient subgraphs.

Rare Disease Inheritance Rare Disease Inheritance uses the same biomedical knowledge graph schema but a different set of patient samples and labels. Each patient subgraph again contains observed positive phenotypes and known causal gene node, while disease nodes are excluded from the target subgraph. The task predicts one or more of seven inheritance patterns from 7,007 patient subgraphs. Among these samples, 1,150 are multi-label.

Protein Set Function Protein Set Function is a protein-set functional classification task. Each sample is a set of proteins curated as participants in the same biological event. The task is to predict the broad functional class represented by that protein set, such as cell cycle, DNA repair, immune system, or signal transduction. The base graph contains protein nodes and pathway-concept nodes. Pathway-concept nodes represent known biological processes and provide background functional context through protein–process associations and process hierarchy (Milacic et al., 2024). The model infers the functional class from the member-protein set and the remaining biological context. The dataset contains 2,034 protein-set subgraphs over 18 classes.

Gene Set Process Gene Set Process is a biological-process classification task over sets of genes and proteins. Each sample is a set of human genes and proteins annotated to the same biological-process term. The task is to predict the broad biological-process category represented by that annotated set. The annotated objects are represented as gene and protein nodes according to their source object type. The base graph also contains biological-process concept nodes. These biological-process nodes represent known process concepts and provide background functional context through gene–process, protein–process, and process-hierarchy relations. The model infers the process category from the annotated set and the remaining ontology context. The dataset contains 4,717 annotated-set subgraphs over nine classes.

Initial Node Features Initial node features are learned once for each base graph and saved for reuse by all dataset variants derived from that graph. CTAP and all comparison methods use the same saved initial node features for each dataset. For DBLP and Movie, a GCN encoder is trained by predicting links in the base graph and its output embeddings are used as node features. For the rare-disease knowledge graph, a GATv2 (Brody et al., 2022) encoder is trained with a DistMult (Yang et al., 2015) objective for predicting relations over the typed knowledge-graph edges. The resulting embeddings are shared by the category and inheritance datasets. For Protein Set Function and Gene Set Process, the same pretraining pipeline is applied separately to each constructed base graph, and the resulting node embeddings are used as initial node features for the corresponding dataset.

F.2 Construction of the Datasets

This section describes how the six datasets in Section 6.1 were constructed. All subgraphs retain the global node identifiers of their base graph. We use a fixed random seed of 3579 for selecting samples and splitting the data unless stated otherwise.

DBLP We construct DBLP from preprocessed bibliographic records containing paper–author, paper–term, and paper–venue associations (Ley, 2002). A venue is mapped to one of 21 target classes, while venue nodes and paper–venue edges are removed from the model input to prevent direct access to the label. The base graph therefore contains paper–author and paper–term edges among 37,472 nodes. Each labeled paper is assigned to one of two views. An author-view sample contains the paper's author nodes, whereas a term-view sample contains its term nodes. We select 10,000 samples for each view while preferring larger node sets and minimizing single-node samples.

Movie We build Movie from the TMDB metadata, cast, and keyword files distributed in Kaggle's *The Movies Dataset* (The Movie Database, 2026; Banik, 2017). Records are joined by the TMDB movie identifier, and movies without actors, keywords, or genre labels are discarded. The heterogeneous graph contains movie, actor, keyword, and genre nodes. Its input edges connect movies to actors and keywords. Genre edges are excluded to prevent label leakage. Each movie is assigned to either the actor view or the keyword view. The assignment is balanced at 10,000 samples per view while favoring the view that provides the richer node set for a movie. Each sample is labeled with all genres associated with that movie.

Rare Disease Category and Rare Disease Inheritance The two rare-disease datasets are constructed from the simulated and MyGene2 patient records accompanying the biomedical knowledge graph used by SHEPHERD (Alsentzer et al., 2025; MyGene2). We use version 8.9.21 of this knowledge graph (KG) as the shared base graph. It contains 105,220 nodes of seven types and 1,620,364 edges. Positive HPO phenotypes and Ensembl causal genes in each patient record are mapped to their corresponding graph nodes (Köhler et al., 2021). A patient sample is retained only when at

least one phenotype, a known causal gene, and a target label can be mapped. The resulting subgraph consists of the positive phenotype nodes and the causal gene node. For Rare Disease Category, target labels are obtained from Orphanet disease categories (Rath et al., 2012). Categories represented by fewer than 500 raw samples are merged into an *Other rare disease category* class. We further control class imbalance and extreme subgraph sizes by downsampling the simulated-patient pool and trimming overrepresented categories, while preserving available MyGene2 samples. This produces 14,482 samples over 13 classes. For Rare Disease Inheritance, target labels are obtained from Orphanet inheritance annotations (Rath et al., 2012). Labels indicating unknown or unavailable information are removed, and inheritance types represented by fewer than 500 raw samples are merged into an *Other inheritance type* class. The simulated-patient pool is capped by its primary inheritance label, and very small or disproportionately large subgraphs are filtered from overrepresented classes. This produces 7,007 samples over seven inheritance classes. Patients may have multiple inheritance labels.

Protein Set Function Protein Set Function is constructed from Reactome human pathway files (Milacic et al., 2024). Reactome pathway entries describe curated biological events and the UniProt proteins that participate in them. Each sample candidate is represented by the member-protein set associated with one such event. The class label is the top-level functional class of the event. Reactome pathway-concept nodes are retained as background context for entries that are neither targets nor labels. Pathway nodes corresponding to the target pathway and top-level label pathways are excluded from the input graph, and direct input edges that would reveal the label are removed. The resulting input to the model is therefore the member-protein set together with the remaining heterogeneous biological context. Pathways with fewer than five or more than 300 member proteins are filtered, and labels with too few samples are removed. The final dataset contains 2,034 samples over 18 classes.

Gene Set Process Gene Set Process is constructed from Gene Ontology and GOA human annotations (Aleksander et al., 2023; Barrell et al., 2009). GO biological-process terms describe known biological processes, and GOA annotations specify which human gene or protein objects are associated with each process. Each sample candidate is represented by the annotated gene/protein set associated with one such process. GOA protein and protein-complex objects are represented as protein nodes, while other GOA gene-product objects are represented as gene nodes. The class label is a broad biological-process ancestor under GO:0008150. Biological-process concept nodes are retained as background context for terms that are neither targets nor labels. Sample GO term nodes and broad label nodes are excluded from the input graph, and direct edges to those label nodes are removed. The resulting input to the model is therefore the annotated gene/protein set together with the remaining GO context. Terms with fewer than five or more than 300 annotated genes/proteins are filtered, and labels with too few samples are removed. The final dataset contains 4,717 samples over nine classes.

F.3 Gene Information and Labels in the Rare Disease Datasets

The MyGene2 cohort used by SHEPHERD retains records with an OMIM disease identifier and a single candidate gene, whereas the simulated cohort is generated from disease-associated phenotypes and genes (Alsentzer et al., 2025). These sources should not be interpreted as exhaustive measurements of each patient's genetic variation. SHEPHERD reports 55 diseases and 48 distinct causal genes in its MyGene2 cohort, so a causal gene does not uniquely identify a disease across that cohort (Alsentzer et al., 2025). However, this does not establish whether gene information alone suffices for our broader disease categories or inheritance labels. Whether the causal gene's pretrained representation is sufficient to predict these labels is therefore an empirical question.

G Additional Experiments

G.1 Experiments on Backbone Variants of Base Methods

Table 8 compares backbone variants for GNNseg, GNNplain, and GLASS. For GNNseg, replacing GCN with an HGNN backbone consistently improves performance on all six datasets. Among the GNNseg variants, SeHGNN performs best overall. For GNNplain, the effect is less consistent.

Table 8: Test micro-F1 of GNNseg, GNNplain, and GLASS variants over ten repeated runs. Values are reported as mean ± standard deviation. Bold indicates the highest value, and underline indicates the second-highest value within each method block for each dataset.

Model	DBLP	Movie	Rare Disease Category	Rare Disease Inheritance	Protein Set Function	Gene Set Process
GCNseg	0.111 ± 0.002	0.042 ± 0.026	0.617 ± 0.036	0.549 ± 0.030	0.309 ± 0.034	0.341 ± 0.021
Simple-HGNseg	$\underline{0.114 \pm 0.005}$	$\underline{0.207 \pm 0.012}$	$\underline{0.678 \pm 0.019}$	$\underline{0.679 \pm 0.014}$	$\mathbf{0.392 \pm 0.065}$	$\underline{0.459 \pm 0.009}$
SeHGNNseg	$\mathbf{0.167 \pm 0.026}$	$\mathbf{0.220 \pm 0.030}$	$\mathbf{0.885 \pm 0.007}$	$\mathbf{0.739 \pm 0.196}$	$\underline{0.330 \pm 0.035}$	$\mathbf{0.489 \pm 0.008}$
GCNplain	$\underline{0.110 \pm 0.002}$	$\underline{0.197 \pm 0.041}$	$\underline{0.510 \pm 0.040}$	$\underline{0.299 \pm 0.037}$	$\underline{0.453 \pm 0.015}$	0.342 ± 0.016
Simple-HGNplain	$\underline{0.110 \pm 0.004}$	$\mathbf{0.263 \pm 0.013}$	0.397 ± 0.035	0.033 ± 0.100	0.400 ± 0.034	$\underline{0.441 \pm 0.003}$
SeHGNNplain	$\mathbf{0.120 \pm 0.008}$	0.189 ± 0.050	$\mathbf{0.771 \pm 0.019}$	$\mathbf{0.307 \pm 0.053}$	$\mathbf{0.470 \pm 0.037}$	$\mathbf{0.480 \pm 0.007}$
GLASS	0.109 ± 0.006	$\mathbf{0.244 \pm 0.047}$	0.176 ± 0.012	$\underline{0.242 \pm 0.067}$	0.274 ± 0.018	0.368 ± 0.022
GLASS + Simple-HGN	$\underline{0.111 \pm 0.002}$	0.148 ± 0.066	$\underline{0.446 \pm 0.054}$	0.234 ± 0.102	$\mathbf{0.425 \pm 0.033}$	$\underline{0.445 \pm 0.005}$
GLASS + SeHGNN	$\mathbf{0.131 \pm 0.009}$	$\underline{0.219 \pm 0.058}$	$\mathbf{0.707 \pm 0.139}$	$\mathbf{0.828 \pm 0.020}$	$\underline{0.419 \pm 0.027}$	$\mathbf{0.510 \pm 0.008}$

Table 9: Test micro-F1 of GNNseg, GNNplain, and GLASS variants with HINPool over ten repeated runs. Values are reported as mean ± standard deviation.

Model (+ HINPool)	DBLP	Movie	Rare Disease Category	Rare Disease Inheritance	Protein Set Function	Gene Set Process
Simple-HGNseg	0.114 ± 0.002	0.140 ± 0.031	0.714 ± 0.019	0.099 ± 0.127	0.359 ± 0.079	0.447 ± 0.002
SeHGNNseg	0.171 ± 0.029	0.121 ± 0.023	0.924 ± 0.012	0.889 ± 0.014	0.474 ± 0.098	0.493 ± 0.008
Simple-HGNplain	0.114 ± 0.004	0.176 ± 0.067	0.548 ± 0.043	0.193 ± 0.101	0.230 ± 0.008	0.447 ± 0.006
SeHGNNplain	0.139 ± 0.027	0.202 ± 0.056	0.878 ± 0.022	0.701 ± 0.217	0.328 ± 0.028	0.465 ± 0.010
GLASS + Simple-HGN	0.116 ± 0.005	0.176 ± 0.053	0.558 ± 0.031	0.256 ± 0.077	0.357 ± 0.095	0.444 ± 0.003
GLASS + SeHGNN	0.120 ± 0.004	0.156 ± 0.061	0.634 ± 0.122	0.867 ± 0.008	0.321 ± 0.024	0.499 ± 0.010

SeHGNN improves five datasets but degrades Movie. Simple-HGN is strongest only on Movie and is weaker than GCN on Rare Disease Category, Rare Disease Inheritance, and Protein Set Function. For GLASS, using SeHGNN gives the strongest variant on four datasets. The original GLASS is strongest on Movie, while Simple-HGN is strongest on Protein Set Function and improves the original GLASS on four datasets. Overall, these results show that heterogeneous node and relation information can improve subgraph-level prediction, but the gains from replacing the backbone are limited.

EIGNNplain achieves 0.972 ± 0.005 on Rare Disease Category and 0.900 ± 0.019 on Rare Disease Inheritance, supporting the importance of long-range dependencies in these tasks. This interpretation is consistent with the substantial same-type attention observed in Section 6.4. In contrast, Simple-HGNplain uses a limited number of layers and neighborhood sampling, which can restrict access to distant information and help explain its low performance on these tasks. On Inheritance, positive-label detection repeatedly failed.

We further apply HINPool to the HGNN-based variants to examine whether heterogeneous pooling provides additional gains beyond replacing the backbone. Comparing Table 9 with Table 8, the effect varies across methods and datasets. SeHGNNseg improves on five datasets but declines on Movie, while GLASS + SeHGNN improves only on Rare Disease Inheritance. Some gains are substantial: SeHGNNplain improves from 0.307 to 0.701 on Rare Disease Inheritance. However, performance also declines in several settings, such as Simple-HGNseg on the same dataset, from 0.679 to 0.099. Thus, adding HINPool can improve particular settings, but does not provide consistent further gains across the HGNN-based baselines.

Simple-HGNseg computes node representations within each target, and HINPool pools them separately by type. Separating these representations at pooling differs from representing each set within a base graph also decomposed by type. Gene and phenotype sets may convey the same disease characteristics despite their different neighborhoods. Combining these representations inappropriately may hinder prediction, offering a possible explanation for the drop from 0.679 to 0.099 on Inheritance and the failure to improve positive-label detection.

Model	DBLP w/o Paper	DBLP with Paper	Movie w/o Movie	Movie with Movie
GCNseg	0.111 ± 0.002	0.109 ± 0.001	0.042 ± 0.026	0.033 ± 0.019
Simple-HGNseg	0.114 ± 0.005	0.115 ± 0.003	0.207 ± 0.012	0.206 ± 0.030
SeHGNNseg	0.167 ± 0.026	0.172 ± 0.014	0.220 ± 0.030	0.176 ± 0.068
S2N+0	0.112 ± 0.003	0.114 ± 0.004	0.256 ± 0.037	0.254 ± 0.038
S2N+A	0.110 ± 0.005	0.113 ± 0.003	0.215 ± 0.050	0.225 ± 0.058
SubGNN	0.285 ± 0.009	0.248 ± 0.011	0.477 ± 0.005	0.491 ± 0.007

Table 10: Results of adding the paper or movie entity to DBLP and Movie target subgraphs. Values are reported as mean $\pm$ standard deviation of test micro-F1.

Table 11: Test micro-F1 of S2N variants over ten repeated runs. Values are reported as mean $\pm$ standard deviation. Bold indicates the highest value in each dataset, and underline indicates the second-highest value.

Model	DBLP	Movie	Rare Disease Category	Rare Disease Inheritance	Protein Set Function	Gene Set Process
S2N+0	$\underline{0.112 \pm 0.003}$	$\mathbf{0.256 \pm 0.037}$	0.169 ± 0.060	$\underline{0.307 \pm 0.040}$	0.255 ± 0.018	0.422 ± 0.048
S2N+A	0.110 ± 0.005	0.215 ± 0.050	$\mathbf{0.266 \pm 0.040}$	0.289 ± 0.038	$\underline{0.301 \pm 0.032}$	$\mathbf{0.448 \pm 0.013}$
CoS2N+0	$\underline{0.112 \pm 0.003}$	$\underline{0.249 \pm 0.055}$	0.141 ± 0.036	$\mathbf{0.327 \pm 0.040}$	0.267 ± 0.014	$\underline{0.431 \pm 0.042}$
CoS2N+A	$\mathbf{0.119 \pm 0.007}$	0.191 ± 0.057	$\underline{0.235 \pm 0.044}$	0.279 ± 0.052	$\mathbf{0.305 \pm 0.033}$	0.429 ± 0.043

CTAP instead decomposes both the target and the base graph, representing gene and phenotype sets through relationships within their respective subproblems before combining their representations. Table 2 evaluates this decomposition; Section 6.4 examines attention by node type.

G.2 Experiments with Inclusion of Paper and Movie Entities

Under the default construction, the target subgraphs of the DBLP and Movie datasets are typed node sets without internal edges. However, GNNseg performs message passing within each subgraph, and S2N+A (Kim & Oh, 2024) uses its internal adjacency. To examine whether such internal structure is useful, we add the corresponding paper entity to each DBLP target subgraph and the corresponding movie entity to each Movie target subgraph, thereby creating internal edges for models that use the structure of each subgraph. For SubGNN (Alsentzer et al., 2020), these edges merge previously separate components into a single connected component, changing the component structure through which anchor-patch information is aggregated. Table 10 reports this comparison for GNNseg, S2N (Kim & Oh, 2024), and SubGNN.

Adding the paper or movie entity does not consistently improve GNNseg. With GCNseg, both DBLP and Movie decrease, and with SeHGNNseg, Movie drops from 0.220 to 0.176 while DBLP increases only marginally from 0.167 to 0.172. The S2N results show a similar pattern. Although S2N+A improves on Movie when the movie entity is added, it remains below S2N+0 on the same variant. On DBLP, S2N+A is also lower than S2N+0 both with and without the paper entity. SubGNN decreases from 0.285 to 0.248 on DBLP, although it increases from 0.477 to 0.491 on Movie. Thus, joining separate components into one connected component does not consistently make anchor-based structural similarity more useful. These results suggest that the internal connectivity among the nodes of a subgraph is not by itself sufficient on these datasets, even for models designed to use that connectivity.

G.3 Experiments on S2N variants

Table 11 compares the four S2N variants. No single variant is dominant across the six datasets. S2N+0 is strongest on Movie, S2N+A is strongest on Rare Disease Category and Gene Set Process, CoS2N+0 is strongest on Rare Disease Inheritance, and CoS2N+A is strongest on DBLP and Protein Set Function. Adding the internal subgraph adjacency improves performance on three datasets for both S2N and CoS2N, but reduces it on the remaining datasets. Coarsening is similarly inconsistent: its effect changes with both the dataset and whether the internal subgraph adjacency is used.

G.4 Experiments on EIGNNplain variants

Table 12 compares the three EIGNNplain formulations. EIGNNplain already outperforms most of the existing subgraph representation methods in Table 13, showing that the long-range dependencies captured by EIGNN provide a strong base-graph signal for subgraph representation. EIGNNplain

Table 12: Test micro-F1 of three EIGNNplain formulations over ten repeated runs. Values are reported as mean ± standard deviation. Bold indicates the highest value, and underline indicates the second-highest value in each dataset.

Model	DBLP	Movie	Rare Disease Category	Rare Disease Inheritance	Protein Set Function	Gene Set Process
EIGNNplain	$\mathbf{0.286 \pm 0.007}$	$\mathbf{0.387 \pm 0.022}$	$\underline{0.972 \pm 0.005}$	$\underline{0.900 \pm 0.019}$	$\underline{0.740 \pm 0.017}$	0.592 ± 0.013
EIGNNplain (heterogeneous)	$\underline{0.285 \pm 0.006}$	0.289 ± 0.090	0.925 ± 0.009	0.456 ± 0.119	0.705 ± 0.013	$\mathbf{0.600 \pm 0.013}$
EIGNNplain (metapath)	0.267 ± 0.052	$\underline{0.381 \pm 0.093}$	$\mathbf{0.974 \pm 0.003}$	$\mathbf{0.912 \pm 0.003}$	$\mathbf{0.745 \pm 0.016}$	$\underline{0.595 \pm 0.011}$

Table 13: Test micro-F1 of existing subgraph representation methods over ten repeated runs. Values are reported as mean ± standard deviation. Bold indicates the highest value in each dataset, and underline indicates the second-highest value.

Model	DBLP	Movie	Rare Disease Category	Rare Disease Inheritance	Protein Set Function	Gene Set Process
GLASS	0.109 ± 0.006	0.244 ± 0.047	0.176 ± 0.012	0.242 ± 0.067	0.274 ± 0.018	0.368 ± 0.022
SSNP	0.122 ± 0.003	0.275 ± 0.030	0.233 ± 0.021	0.275 ± 0.062	0.657 ± 0.016	0.525 ± 0.015
S2N+A	0.110 ± 0.005	0.215 ± 0.050	0.266 ± 0.040	0.289 ± 0.038	0.301 ± 0.032	0.448 ± 0.013
ISNN	0.095 ± 0.013	0.182 ± 0.101	0.085 ± 0.016	0.223 ± 0.061	0.244 ± 0.028	0.342 ± 0.057
GATplain	0.118 ± 0.001	0.167 ± 0.008	0.375 ± 0.155	0.360 ± 0.002	0.254 ± 0.041	0.445 ± 0.016
GPEN	0.243 ± 0.014	$\underline{0.443 \pm 0.017}$	0.757 ± 0.039	0.736 ± 0.027	0.626 ± 0.045	$\underline{0.555 \pm 0.014}$
Rad et al.	0.262 ± 0.009	0.366 ± 0.028	$\mathbf{0.950 \pm 0.009}$	0.833 ± 0.045	0.712 ± 0.036	0.473 ± 0.034
SubGNN	$\underline{0.285 \pm 0.009}$	$\mathbf{0.477 \pm 0.005}$	$\underline{0.932 \pm 0.010}$	$\mathbf{0.865 \pm 0.008}$	$\mathbf{0.813 \pm 0.041}$	0.548 ± 0.094
WLKS-$\{0, D\}$	$\mathbf{0.293 \pm 0.000}$	0.432 ± 0.000	0.930 ± 0.000	$\underline{0.861 \pm 0.000}$	$\underline{0.762 \pm 0.000}$	$\mathbf{0.601 \pm 0.000}$

(heterogeneous) is weaker than EIGNNplain on five of the six datasets and improves only Gene Set Process. EIGNNplain (metapath) outperforms EIGNNplain on Rare Disease Category, Rare Disease Inheritance, Protein Set Function, and Gene Set Process, while EIGNNplain remains stronger on DBLP and Movie. These results show that type information organized through appropriate metapath semantics can complement the long-range dependencies captured by EIGNN alone.

G.5 Existing Subgraph Representation Methods

Table 13 compares existing subgraph representation methods and the GATplain baseline on the six datasets. WLKS-$\{0, D\}$ (Kim & Oh, 2025) leads on DBLP and Gene Set Process and performs close to SubGNN on the rare disease tasks, but trails it more substantially on Movie and Protein Set Function. This contrast suggests that the relative effectiveness of WLKS and anchor-based methods varies across tasks. GPEN captures relationships between distant nodes through encoding of global positions (Wu et al., 2025). Its strong performance can be considered alongside the EIGNNplain results in Table 12, since implicit graph neural networks offer a way to capture long-range dependencies beyond a fixed number of message-passing layers (Liu et al., 2021; Zhong et al., 2025). Together, these observations suggest that accessing information beyond local node neighborhoods is useful for the evaluated subgraph-level tasks.

SubGNN and Rad et al. share a mechanism that sends messages directly from anchor patches to subgraph components (Alsentzer et al., 2020; Hamidi Rad et al., 2023). This provides a route for anchor information to reach a target component without successive node-level propagation along the intervening graph paths, potentially reducing the difficulty of accessing distant information. The strong results of both methods are consistent with the effectiveness of this shared mechanism. However, Rad et al. outperform SubGNN only on Rare Disease Category, while SubGNN performs better on all five remaining datasets. Thus, selecting anchors through metapaths to incorporate heterogeneous information does not generally translate into performance gains over the shared SubGNN mechanism in these experiments.

GATplain, GLASS, and S2N+A remain below GPEN, Rad et al., and SubGNN on all six datasets. SSNP is competitive on the biological member-set tasks, but is weaker than these three methods on the other datasets. ISNN's low performance can be interpreted in terms of the utility of the additional connections between subgraph nodes. Whereas EIGNNplain aggregates node representations propagated over the base graph, ISNN also receives information from other targets through

Table 14: Two-sided Welch's unpaired t-test p-values comparing CTAP with each baseline over ten runs per method. Bold identifies the comparison with the highest-mean baseline in each dataset. Values are unadjusted for multiple comparisons.

Baseline	DBLP	Movie	Rare Disease Category	Rare Disease Inheritance	Protein Set Function	Gene Set Process
SeHGNNseg + HINPool	1.190×10^{-7}	1.307×10^{-15}	1.417×10^{-7}	3.867×10^{-6}	9.203×10^{-7}	2.365×10^{-15}
EIGNNplain	4.196×10^{-4}	5.954×10^{-12}	0.098	3.561×10^{-4}	1.578×10^{-6}	2.513×10^{-4}
EIGNNplain (metapath)	0.086	9.405×10^{-5}	**0.022**	$\mathbf{5.112 \times 10^{-12}}$	3.184×10^{-6}	2.631×10^{-4}
GLASS + SeHGNN	7.287×10^{-19}	8.088×10^{-9}	3.205×10^{-4}	5.945×10^{-8}	3.911×10^{-17}	1.559×10^{-14}
GPEN	2.685×10^{-8}	1.635×10^{-12}	2.852×10^{-8}	3.159×10^{-9}	1.602×10^{-8}	6.161×10^{-9}
SubGNN	1.382×10^{-3}	$\mathbf{3.013 \times 10^{-14}}$	1.636×10^{-7}	8.850×10^{-11}	**0.637**	0.052
Rad et al.	1.120×10^{-8}	2.216×10^{-10}	4.228×10^{-4}	8.438×10^{-5}	1.571×10^{-6}	1.367×10^{-7}
WLKS-$\{0, D\}$	**0.019**	2.655×10^{-12}	1.004×10^{-7}	1.234×10^{-14}	1.069×10^{-4}	$\mathbf{9.562 \times 10^{-4}}$

these connections. When the connected targets provide information unrelated to the prediction, the additional connections can mix signals from different targets and reduce their distinguishability.

ISNN uses label information to connect training subgraphs, whereas an evaluation subgraph is connected to a training subgraph with a similar pretrained representation. If representation similarity does not reflect label-relevant relationships, information transfer that was useful during training can instead introduce noise during evaluation. These connections are fixed after construction from the pretrained representations, so subsequent classifier training does not correct the choice of connected targets. Consequently, even with an implicit backbone that handles long-range dependencies, uninformative additional connections can lead to lower performance than EIGNNplain, which uses the original base-graph relations.

G.6 Welch's t-Tests for the Main Results

For each dataset, we compare the ten test micro-F1 scores of CTAP with those of each baseline in Table 1 using a two-sided Welch's unpaired t-test (Welch, 1947). The test does not assume equal population variances or pair runs by their stored positions. The test statistic is $t = (\bar{x} - \bar{y})/\sqrt{s_x^2/n_x + s_y^2/n_y}$, where $\bar{x}, \bar{y}$ and s_x^2, s_y^2 are the sample means and variances for CTAP and the baseline, respectively, and $n_x = n_y = 10$. The approximate degrees of freedom are

$$\nu = \frac{(s_x^2/n_x + s_y^2/n_y)^2}{(s_x^2/n_x)^2/(n_x - 1) + (s_y^2/n_y)^2/(n_y - 1)}.$$

The null hypothesis is equality of the two population means. Table 14 reports the unadjusted p-values.

Each p-value compares one baseline with CTAP on one dataset at a significance threshold of 0.01. Because the test accounts for variation within each method's runs, a larger mean gap need not yield a smaller p-value; the highest-mean baseline need not have the largest p-value.

Table 14 shows that every baseline comparison on Movie and Rare Disease Inheritance yields $p < 0.01$. Comparisons with SeHGNNseg + HINPool, GLASS + SeHGNN, GPEN, and Rad et al. meet this threshold on all six datasets. In contrast, comparisons with EIGNNplain on Rare Disease Category ($p = 0.098$), EIGNNplain (metapath) on DBLP ($p = 0.086$) and Rare Disease Category ($p = 0.022$), SubGNN on Protein Set Function ($p = 0.637$) and Gene Set Process ($p = 0.052$), and WLKS-$\{0, D\}$ on DBLP ($p = 0.019$), are not significant at the 1% level. These nonsignificant results do not establish equal performance.

G.7 Inference Cost of Semantic Fusion

Table 15 compares inference time with SF, mean aggregation, and concatenation on Movie and Rare Disease Category.[1] We fix each dataset's selected CTAP configuration and replace only the fusion

[1]These timings were measured separately from Table 3, without inference warmup; Table 3 uses one warmup pass for CTAP.

Table 15: Inference time (seconds) for the complete test split under a fixed CTAP configuration.

Fusion	Movie	RD Category
SF	0.2154	0.5561
Mean	0.1653	0.4844
Concat	0.1658	0.4850

Table 16: Testing whether gene information suffices for rare-disease prediction. Test micro-F1 is reported as mean ± standard deviation over ten runs.

Condition	Category	Inheritance
Initial features — gene	0.240 ± 0.011	0.252 ± 0.055
Initial features — phenotype	0.155 ± 0.015	0.219 ± 0.086
Initial features — gene + phenotype	0.223 ± 0.017	0.216 ± 0.099
GCNplain — gene	0.718 ± 0.021	0.305 ± 0.102
GCNplain — gene + phenotype	0.510 ± 0.040	0.299 ± 0.037

modules, as described in Appendix K.1. Each measurement covers one prediction pass over the complete test split after one training epoch, with CUDA synchronization and no inference warmup. Data loading and preprocessing are excluded. These measurements isolate the inference-cost difference under a common configuration; the performance results in Table 2 use separately tuned variants.

All three variants complete test inference in under 0.6 s. In Table 2, SF achieves higher mean performance than mean aggregation on all six datasets and than concatenation on five, although concatenation improves Category by 0.2 pp. Together, these results indicate a performance–inference-time trade-off.

G.8 Assessing Trivial Prediction and Label Leakage in Rare Disease Tasks

Known causal genes, gene–disease edges, and KG-pretrained features raise the concern that rare-disease labels may be nearly determined by the inputs. Table 16 tests whether initial gene features or gene representations after KG propagation suffice for prediction. Gene provenance is described in Appendix F.3.

All conditions use the complete datasets and existing splits without additional filtering; every sample contains one causal gene and positive phenotypes. The first three use fixed initial features without graph propagation: gene features, mean-pooled phenotype features, or their concatenation. Their prediction head has a 64-dimensional hidden layer, ReLU, and dropout. The fourth applies GCNplain on the full base graph, including gene–disease edges, and reads the patient's gene embedding. These four conditions each use 50 Optuna trials selected by validation micro-F1 and ten evaluation runs. The fifth reuses existing GCNplain results with gene and phenotype pooling.

Initial gene features outperform phenotype features but reach only 0.240 on Category and 0.252 on Inheritance; concatenating phenotype features does not improve either mean. Although KG propagation makes gene information more predictive, raising gene-only scores to 0.718 and 0.305, a single-gene readout cannot exploit the target subgraph's node composition and relations as subgraph representation methods do. The substantial attention to translated views at gene/protein nodes in Section 6.4 illustrates CTAP's use of such relations. The gene-only results therefore do not account for the high performance of those methods. Gene-only GCNplain exceeds gene + phenotype pooling on Category and differs by just 0.6 pp on Inheritance, with greater variation; these results do not establish a benefit from adding phenotypes to GCNplain's pooling.

Thus, the tested predictors do not nearly recover the labels from gene information, even with KG propagation. This argues against assuming trivial prediction from the availability of causal genes, gene–disease edges, and pretrained features alone, although it does not establish the absence of label leakage.

H IMPLEMENTATION OF BASELINE VARIANTS

SubGNN (Alsentzer et al., 2020) and Rad et al.'s extension (Hamidi Rad et al., 2023) pass messages from sampled anchor patches at the component level.

H.1 HGNN IMPLEMENTATIONS OF GNNSEG AND GNNPLAIN

We name each variant by its backbone followed by seg or plain. Simple-HGN (Lv et al., 2021) and SeHGNN (Yang et al., 2023) backbones test heterogeneity handling in GNNseg and GNNplain against GCN (Kipf & Welling, 2017).

Simple-HGNseg and SeHGNNseg These variants compute node embeddings within each target subgraph and pool them to obtain its representation. SeHGNNseg aggregates features along predefined metapaths within the target subgraph and supplies them to SeHGNN (Yang et al., 2023). Simple-HGNseg uses one-hop edge attention with edge-type embeddings (Lv et al., 2021).

SeHGNNplain This variant applies SeHGNN on the base graph and pools the embeddings of the nodes in the target subgraph. Features are aggregated along the predefined metapaths on the full base graph and saved for reuse by SeHGNN.

Simple-HGNplain This variant applies Simple-HGN on the base graph and pools the embeddings of the nodes in the target subgraph. To reduce GPU memory use, we sample neighbors using the method introduced in the GraphSAGE paper (Hamilton et al., 2017). Sampling starts from the nodes in each target subgraph and extends up to k hops, where k is the number of message-passing layers. This follows the definition of a subgraph neighborhood in WLKS (Kim & Oh, 2025). We sample up to 15 neighbors per node at the first hop, 10 at the second, and 5 at each remaining hop.

GATplain GATplain tests the effectiveness of the GAT-based mechanism used in SHEPHERD (Alsentzer et al., 2025). This variant uses a GATv2 (Brody et al., 2022) backbone for message passing on the shared base graph and pools the embeddings of the nodes belonging to each target subgraph. It uses the same neighbor sampling as Simple-HGNplain, with k equal to the number of GATv2 layers.

H.2 GLASS INSTANTIATIONS WITH HGNN BACKBONES

We incorporate the zero-one subgraph membership labels of GLASS (Wang & Zhang, 2022) into SeHGNN (Yang et al., 2023) and Simple-HGN (Lv et al., 2021), retaining their metapaths and node and edge types, respectively.

Metapath-Based Instantiation For SeHGNN, we retain each predefined metapath and apply membership labels only to its source and target nodes. We do not separately label the intermediate nodes. Let $\mathbf{z} \in \{0,1\}^{|\mathcal{V}|}$ denote the zero-one membership labels, where $z_u = 1$ if node u belongs to the target subgraph and $z_u = 0$ otherwise. For a row-normalized metapath transition matrix $\widehat{\mathbf{A}}_p$, we separately aggregate features from source nodes inside and outside the target subgraph:

$$\mathbf{X}_p^L = \widehat{\mathbf{A}}_p \left(\mathbf{z} \odot \mathbf{X}\right), \qquad \mathbf{X}_p^G = \widehat{\mathbf{A}}_p \left((\mathbf{1} - \mathbf{z}) \odot \mathbf{X}\right). \tag{42}$$

Here, $\odot$ denotes element-wise multiplication with the membership vector broadcast across feature columns. The local and global aggregated features are projected with separate metapath-specific MLPs, mixed with a soft sharing ratio, and passed to SeHGNN's semantic fusion module.

Metapath-Free Instantiation For Simple-HGN, membership labels modify only the attention scores. The node and edge types, message projection, and aggregation remain unchanged. For each edge $u \to v$, we compute a global attention logit e_{uv}^G and a subgraph-local attention logit e_{uv}^L for the same typed relation. Using the source membership z_u and a mixing ratio $\alpha \in [0,1]$, the final logit is

$$\widetilde{e}_{uv} = z_u \left[\alpha e_{uv}^L + (1-\alpha) e_{uv}^G\right] + (1 - z_u) \left[\alpha e_{uv}^G + (1-\alpha) e_{uv}^L\right]. \tag{43}$$

For source nodes in the target subgraph, the local and global scores receive weights α and $1-\alpha$; for other source nodes, these weights are reversed. The mixed logits are normalized once over the neighborhood, and Simple-HGN's original attention residual, aggregation of messages, and normalization of outputs are retained.

H.3 EIGNNplain Variants

EIGNNplain (Liu et al., 2021) and its heterogeneous and metapath variants test improved modeling of long-range dependencies over other GNNplain variants.

EIGNNplain (heterogeneous) uses a separate message-passing function for each relation type, following the heterogeneous extension of IGNN (Gu et al., 2020) and the relation-specific transformations of R-GCN (Schlichtkrull et al., 2018). EIGNNplain (metapath) builds a separate metapath-based neighborhood graph for each predefined metapath, applies EIGNNplain to each neighborhood graph, and semantically fuses the resulting representations.

H.4 Applying HINPool to Baseline Models

We apply HINPool (Hong et al., 2026) to the Simple-HGN and SeHGNN variants of GNNseg, GNNplain, and GLASS by pooling the embeddings of nodes in each target subgraph separately by node type. Within each target subgraph, a learned linear function for each node type followed by a sigmoid scores the nodes of that type. We retain the highest-scoring nodes of each type according to the pooling ratio, rounding the number of retained nodes up and keeping at least one node of each type present in the target subgraph. We multiply the selected embeddings by their scores and take their mean separately for each type. The resulting vectors are concatenated across types and passed to the classification head; a type absent from the target subgraph contributes a zero vector. The embeddings supplied to this pooling are computed by the corresponding GNNseg, GNNplain, or GLASS encoder.

H.5 S2N Variants

Using our notation, S2N (Kim & Oh, 2024) translates each S_i into a node and assigns the raw edge weight

$$\widehat{\mathbf{A}}_{ij} = T_e(S_i, S_j) = \sum_{u \in \mathcal{V}_{S_i}} \sum_{v \in \mathcal{V}_{S_j}} \mathbf{A}_{uv}.$$

These weights are standardized, thresholded, and normalized before message passing. Let $\mathbf{A}_{S_i}$ be the internal adjacency of S_i. Adapting the original feature-encoding formulation to our implementation, the translated node features are

$$\text{S2N+0:} \quad \widehat{\mathbf{x}}_i = \sum_{v \in \mathcal{V}_{S_i}} \mathbf{X}_v,$$

$$\text{S2N+A:} \quad \widehat{\mathbf{x}}_i = \mathrm{Pool}(\mathrm{GCN}_{\mathrm{node}}(\mathbf{X}_{S_i}, \mathbf{A}_{S_i}))\,,$$

where Pool aggregates member-node representations. Stacking these vectors as rows of $\widehat{\mathbf{X}}$, both variants apply a GCN on the translated graph and a linear classification head.

CoS2N adds virtual subgraphs produced by graph coarsening:

$$\mathcal{S}_{\mathrm{co}} = \mathrm{Coarsening}(G), \qquad \widehat{\mathbf{A}}^{\mathrm{co}}_{ij} = T_e(S_i, S_j), \quad S_i, S_j \in \mathcal{S} \cup \mathcal{S}_{\mathrm{co}}.$$

Each virtual subgraph contains the original nodes assigned to a coarse node. CoS2N+0 and CoS2N+A use the corresponding feature encoders above; virtual subgraphs participate in propagation but not in the supervised loss.

I Training and Evaluation

All experiments were conducted on NVIDIA RTX A6000 GPUs with 48 GB of memory each. We tune CTAP and the baselines separately for each dataset using Optuna (Akiba et al., 2019) with a

Table 17: Selected CTAP hyperparameters for the six datasets.

Dataset	Layers	Learning rate	Dropout	Pooling	Batch size	Reduction factor
DBLP	7	5.466×10^{-4}	0.050	mean	4096	0.586
Movie	8	1.674×10^{-3}	0.200	mean	1024	0.336
Rare Disease Category	6	1.105×10^{-3}	0.416	mean	512	0.619
Rare Disease Inheritance	8	5.992×10^{-4}	0.183	mean	256	0.865
Protein Set Function	5	1.261×10^{-3}	0.193	max	128	0.473
Gene Set Process	8	1.126×10^{-4}	0.044	max	256	0.360

budget of 50 trials per model and dataset, selecting hyperparameters by validation micro-F1. For CTAP, we use a Tree-structured Parzen Estimator (TPE) sampler and evaluate each trial by its best validation micro-F1. The search ranges are specified in Appendix J.1.

For CTAP, we train using the Adam optimizer (Kingma & Ba, 2015) and use `ReduceLROnPlateau` on the average training loss, with the searched reduction factor and a minimum learning rate of 5×10^{-5}. Each trial runs for at most 100 epochs. Validation micro-F1 is evaluated every three epochs, and training stops after five evaluations without a strict improvement.

The selected configuration is evaluated in ten repeated runs with the same training, validation, and test splits and different random seeds. Each run initializes the model independently; the seed also controls shuffling of training targets and dropout. For single-label tasks, the predicted class is the one with the highest logit. For multi-label tasks, a label is predicted when its sigmoid probability exceeds 0.5, equivalently when its logit is positive. We report the mean and standard deviation of test micro-F1 over the ten runs.

J Hyperparameter Selection

J.1 Hyperparameter Search

The hidden feature dimension is fixed at 64.

The search range for the number of MPNN layers is $\{1, \ldots, 8\}$. For Rare Disease Category, the upper bound was reduced to seven after the first five completed trials. The search range for the learning rate is $[10^{-4}, 10^{-2}]$ with log-uniform sampling. The search range for the dropout probability is $[0, 0.6]$ with uniform sampling. The search range for the learning-rate reduction factor is $[0.3, 0.9]$ with uniform sampling. The search choices for the subgraph representation used for classification are concatenation of the representations from all layers and the representation from the final layer only. The search range for the minibatch size is $\{1024, 2048, 4096\}$ for DBLP and Movie, $\{512, 1024, 2048, 4096\}$ for Rare Disease Category, $\{256, 512, 1024, 2048\}$ for Rare Disease Inheritance, $\{64, 128, 256, 512\}$ for Protein Set Function, and $\{128, 256, 512, 1024\}$ for Gene Set Process.

The search candidates for $\mathrm{Pool}(\cdot)$ in Section 5.3 are mean pooling, max pooling, and size pooling. Size pooling divides the sum of the input vectors by the square root of their number. The selected function is shared by the pooling operations for cross-type neighbors and target constituent nodes. The Rare Disease Category search also includes one completed trial with sum pooling.

J.2 Selected Hyperparameters

Table 17 lists the selected configuration for each dataset. All six configurations use a hidden dimension of 64 and concatenate subgraph representations from all layers for classification. Batch size denotes the number of training targets per minibatch; the reduction factor controls the learning-rate scheduler. Continuous values are rounded for display.

K Implementation Details of Ablation and Efficiency Experiments

K.1 Experimental Setup for the Ablation Study

Experimental Setup The variants in Table 2 use CTAP's datasets, input features, labels, and splits. Unaffected components remain unchanged. Stage-specific target sets follow Appendix C.11.

Each variant is tuned per dataset by validation micro-F1 over 25–50 trials, using the ranges in Appendix J.1 and the optimizer, stopping, and prediction rules in Appendix I. Selected configurations are trained with ten different seeds; results are mean and standard deviation of test micro-F1. Full CTAP results are those in Table 1.

Removal of Individual View Types Each ablation removes the specified views across all node types and relations, including their MPNNs and fusion inputs. Other views retain their construction and propagation. Fusion and membership pooling use the available representations.

Without node-overlap views Node-overlap message passing is removed. Projected-anchor membership is preserved for pooling, which uses nodes represented by the remaining views.

Without edge-connectivity views Edge-connectivity message passing is removed; base-graph same-type edges remain available to same-type message passing.

Without same-type views Direct same-type message passing is removed, but same-type edges still determine edge-connectivity weights. This includes all types, including pathways in Protein Set Function, so the effect is not specific to protein interactions.

Applicability DBLP and Movie lack same-type and edge-connectivity views; removing node-overlap views would remove all views. Their three view ablations are therefore N/A. View ablations cover the other four datasets, and all remaining ablations cover all six.

Removal of Membership Pooling Cross-type neighbor pooling into node representations and its subsequent semantic fusion are removed. Fused view outputs directly become the next layer's node states.

All views, pooling into target representations, and target fusion remain. This variant therefore tests node-level cross-type pooling, not pooling constituent-node representations into targets.

Removal of Cross-Type Projection Projected-anchor nodes and membership rows are removed. Target nodes, initial features, target-to-target relations, type decomposition, relation-specific propagation, and same-type views remain; target-to-target views remain separate.

Node representations come from same-type views. Cross-type pooling requiring projected-anchor membership is removed, while target pooling uses nodes represented by same-type views. No replacement encoder is added for other types. Thus, projection and its dependent operations are removed together.

Removal of Type Decomposition Ignoring node types produces one target-only S2N node-overlap view. Each target's initial feature sums its nodes' original features, and edge weights count shared nodes. Propagation and normalization follow CTAP.

Projected anchors, edge-connectivity views, same-type views, and membership pooling are absent. Target and final fusion each receive one input. The classifier and the search for layer count and final-layer versus concatenated representations remain unchanged.

This compares a target-only view with CTAP's type-specific construction; projection and view construction also change, so the difference does not isolate type decomposition.

Replacement of Semantic Fusion All semantic fusion modules are replaced with either an unweighted mean over available inputs or concatenation followed by a learned linear projection to the original hidden dimension. Concatenation uses a fixed input order and zero vectors for unavailable inputs. These replacements apply to view fusion, membership-pooling fusion, and final fusion across

target types. View construction, message passing, and membership pooling remain unchanged. Each variant is tuned with 50 trials per dataset and evaluated over ten runs.

K.2 Experimental Setup for the Efficiency Study

Training time is measured over one epoch on the training split, while inference time and memory are measured when predicting the complete test split. CTAP uses its standard evaluation graph containing training and test targets and extracts predictions for test targets only. Timing uses CUDA synchronization and excludes validation from the training epoch. CTAP, GPEN, EIGNNplain (metapath), and SeHGNNseg + HINPool have one training warmup epoch; SubGNN has none. Inference has one warmup pass except for SubGNN on Category. Memory values are the peak allocated GPU memory during inference. These short runs measure computational cost rather than converged-training time.

Preprocessing covers model-specific preparation before training. For CTAP, it includes cross-type projection, construction of the three types of views, and preparation of their training and evaluation structures. For GPEN, it includes spanning-forest construction, positional features, and perturbed target subgraphs. For EIGNNplain (metapath), it includes metapath propagation operators and features, and node-to-subgraph membership matrices; for SeHGNNseg + HINPool, it includes metapath transition matrices and batching structures. SubGNN's preprocessing workflow includes connected-component extraction, anchor-patch sampling, random walks, and structural similarities between components and anchors. Dataset construction and initial node-feature pretraining are outside this comparison.

K.3 Experimental Setup for the View Attention Analysis

We use each dataset's selected CTAP configuration and ten runs, following the main training and model-selection procedure. After restoring the selected model, we collect attention in evaluation mode on the graph containing training and test targets. We measure semantic fusion of view representations for base-graph nodes before membership pooling, excluding membership fusion and final fusion across target types.

For each node and layer, we average attention over available query views to obtain the incoming weight for each key view. The heatmaps average each view's weight over nodes belonging to that view, then equally over layers and runs. View indices follow each dataset's construction order. Since individual views can cover different node populations, heatmap rows need not sum to one.

For the view-type totals discussed in Section 6.4, we sum incoming weights within each view type and average over the observed nodes of the specified node type, followed by equal averaging over layers and runs. For all-node averages, we first combine the weight sums and node counts across node types. These totals are computed from the saved records, rather than by adding the heatmap's individual-view means.

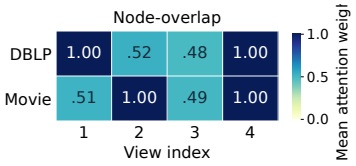

Figure 9: Mean view attention on DBLP and Movie.

DBLP and Movie have only node-overlap views and are therefore shown separately in Figure 9. At movie anchors, the actor-side and keyword-side projection views receive 0.51 and 0.49, respectively. Table 2 and Appendix G.2 examine projection removal and added internal target connectivity, respectively. DBLP's paper anchors similarly assign 0.52 and 0.48 to the author-side and term-side views. Weights of one occur where only one view is available.

Category and Inheritance share the same base graph and show similar attention patterns (Figure 10), motivating the use of Inheritance alone in Figure 5. In Category, the node-overlap, edge-connectivity, and same-type totals are 0.422/0.442/0.136 for gene/protein nodes and 0.223/0.236/0.533 for phenotype nodes. In Inheritance, they are 0.430/0.435/0.135 and 0.235/0.241/0.521, respectively. Thus, the difference in emphasized relations between these node types discussed in Section 6.4 appears in both tasks.

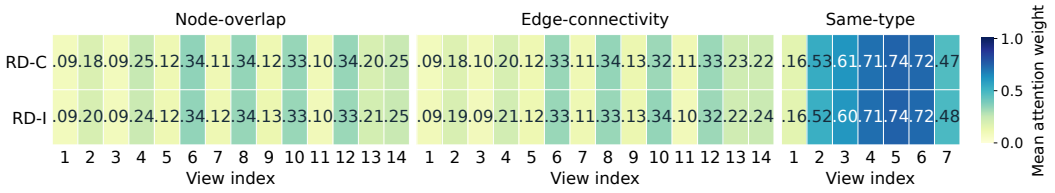

Figure 10: Mean view attention on the two rare disease tasks sharing the same base graph. RD-C and RD-I denote the Rare Disease Category and Rare Disease Inheritance datasets, respectively.

K.4 Experimental Setup for the Layer-Count Analysis

We vary the number of CTAP layers L from 1 to 8 on Movie, Rare Disease Inheritance, and Protein Set Function, using the same datasets, pretrained node features, and training, validation, and test splits as in the main experiments. For each dataset, all hyperparameters other than L are fixed to its selected CTAP configuration in Appendix J.2; we do not retune them for individual layer counts. All three configurations concatenate representations across layers, so increasing L also changes the dimension of the concatenated representation.

Each layer count is evaluated over ten independently initialized runs using seeds $2^i - 1$ for $i = 0, \ldots, 9$. Optimization, validation, stopping, and test-score reporting follow Appendix I: training runs for at most 100 epochs, validation is performed every three epochs, and early stopping uses five evaluations without a strict improvement. Figure 6 reports the mean test micro-F1 and shades one standard deviation over the ten runs at each layer count.

L Limitations

Transductive Setting Our experiments consider a transductive setting in which target subgraphs belong to a given shared base graph. Extending CTAP to inductive settings, including generalization to previously unseen base graphs, is a direction for future research.

Memory Requirements Although CTAP avoids explicitly constructing translated adjacency matrices, it still stores membership matrices and intermediate representations for each view. Memory requirements can therefore increase with the number of views and the total size of target memberships. Improving memory efficiency at larger scales remains an important direction.

Evaluation with Multiple Relations CTAP constructs projections and views separately for each relation, allowing multiple cross-type relations between the same pair of node types. However, our experiments do not evaluate this setting, leaving its performance and efficiency to future work. The simplified complexity bounds based on the number of node types assume at most one cross-type relation per pair of distinct node types; with multiple relations, the bounds retaining the actual relation counts apply instead.

Tasks Beyond Classification Tasks beyond classification remain less explored in subgraph representation learning. Our classification experiments assess whether the learned representations distinguish the supplied labels, but do not establish whether similarities between those representations reflect meaningful relationships between subgraphs beyond these labels. A concrete direction is to formulate tasks for retrieving or grouping related subgraphs, with relevance criteria that account for their node membership and heterogeneous graph context. Such evaluations would help clarify which relationships the representations preserve beyond those required for classification.