END-TO-END SUBGRAPH DETECTION WITH GRAPHDETR

Anonymous authors
Paper under double-blind review

ABSTRACT

Subgraph detection seeks to identify whether and where instances of given patterns occur within a larger graph. This problem is fundamental across scientific domains and is closely related to subgraph isomorphism, which is NP-complete, limiting combinatorial approaches to small patterns or moderately sized graphs. We introduce GraphDETR, a deep learning framework that formulates subgraph detection as a set prediction problem, analogous to DETR in object detection. GraphDETR encodes the target graph with a graph neural network, and employs a fixed set of learnable query vectors, decoded via a transformer decoder, to predict all pattern occurrences jointly in a single forward pass. This is enabled by training the model end-to-end with bipartite matching. Unlike traditional combinatorial methods that only solve exact structural matching, GraphDETR naturally extends to approximate matching, enabling detection beyond exact pattern correspondence. Empirically, we show that GraphDETR can detect diverse patterns, such as molecular structures, cycles, cliques, and fuzzy patterns of up to 50 nodes, in target graphs with up to 1000 nodes. We further evaluate on molecular functional group detection over the ChEMBL dataset, where GraphDETR predicts the complete set of functional groups per molecule, achieving a strong performance of $\mathrm{AP}_{100} = 91.2$.

1 INTRODUCTION

Subgraph detection is a fundamental problem in graph analysis that aims to identify whether and where specific patterns occur within a larger graph. It has applications in many domains such as molecular analysis (Raymond et al., 2002; Yang and Sze, 2007), and network science (Plotnick, 1997; Gentner, 1983). A prominent instance of this problem is molecular functional group detection, where the goal is to locate all occurrences of chemically characteristic substructures such as hydroxyl or carbonyl groups within a molecule. Identifying these substructures plays a key role in tasks such as reaction prediction, retrosynthesis planning, and molecular property modeling. Figure 1A illustrates this task on an example molecule.

Classical approaches to locating pattern occurrences in graphs rely on combinatorial search algorithms for subgraph isomorphism, which seek injective edge-preserving mappings from a pattern to a target graph. Methods such as Ullmann (1976) and Cordella et al. (2004) provide exact solutions but suffer from exponential worst-case complexity, which limits their applicability on large graphs. Recent neural approaches (Ying et al., 2021b) attempt to address this limitation by learning representations of graphs or subgraphs using graph neural networks (GNNs). However, most existing methods decompose the task into local matching decisions or independent predictions of candidate substructures. As a result, they do not explicitly model the fact that a target graph typically contains a *set of subgraphs* that should be predicted jointly while avoiding duplicate detections.

Subgraph detection thus calls for jointly predicting all occurrences in a target graph, each as a labeled set of nodes, without duplicate detections. Casting the problem this way reveals a direct parallel with object detection, where the goal is likewise to predict a variable-sized set of objects, each with a class and a location, without duplicates (Figure 1B). In fact, recent progress in computer vision has shown that object detection can be solved elegantly as set prediction. In particular, DETR (Carion et al., 2020) demonstrated this by training a fixed pool of learnable queries via a transformer decoder with bipartite matching. DETR removes many heuristic components of traditional detection pipelines and enables fully end-to-end training.

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.

In this work, we bring the set prediction perspective to graphs and introduce GraphDETR, an end-to-end framework for subgraph detection. GraphDETR first encodes the target graph using a GNN. A fixed set of learnable query vectors then interacts with the graph representation through a transformer decoder to produce candidate subgraphs. Each query predicts a node-set mask representing a detected subgraph. GraphDETR is trained end-to-end with a bipartite matching loss that assigns predictions to ground-truth subgraphs, producing the complete set of detections in a single forward pass without duplicate predictions. The model operates in a closed-vocabulary setting where pattern classes are fixed at training time rather than provided as input at inference. Unlike combinatorial methods, which are inherently limited to exact structural matching, GraphDETR naturally extends to approximate matching, where a pattern class covers a set of subgraphs rather than a single exact structure.

We evaluate GraphDETR in two settings. First, we apply GraphDETR to molecular functional group detection, where the task is to predict the complete set of functional groups present in a molecular graph. Experiments on the ChEMBL dataset show that GraphDETR achieves strong performance while maintaining a simple, unified, and fully end-to-end architecture. Second, we consider subgraph matching with a fixed query set, which provides a general benchmark for evaluating the model's ability to predict multiple substructures simultaneously. Overall, our approach demonstrates that the set prediction paradigm provides a natural and effective formulation for graph reasoning problems involving multiple structured outputs. By combining GNNs with transformer-based set prediction, GraphDETR provides a new approach for learning to detect subgraphs directly from data.

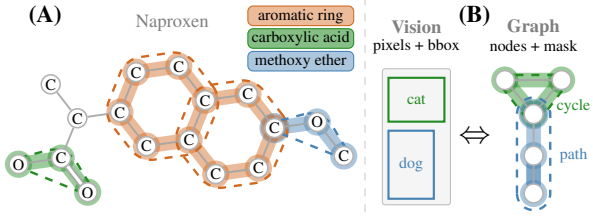


Figure 1: **(A)** A molecular graph with three color-coded functional groups, each predicted as a class label and a binary node mask. **(B)** The analogy between object detection and subgraph detection, where bounding boxes over pixels correspond to node masks over graphs.

2 Related Work

Set prediction. DETR (Carion et al., 2020) reformulated object detection as direct set prediction, where a transformer decoder with a fixed pool of learnable query vectors attends to image features and is trained end-to-end via Hungarian bipartite matching (Kuhn, 1955), eliminating anchors and non-maximum suppression. Subsequent work refined query initialization (Zhang et al., 2022) and extended the paradigm to instance segmentation (Cheng et al., 2022). SAM (Kirillov et al., 2023) and SAM 2 (Ravi et al., 2025) introduced a *two-way* transformer decoder in which prompt tokens and image tokens mutually attend to each other, allowing the image representation to incorporate per-query context before mask logits are computed. GraphDETR applies this full pipeline to graphs, where image feature maps become per-node embeddings produced by a graph encoder, bounding boxes become binary node-membership masks, and class labels become subgraph types. The principal graph-specific addition is the cut penalty, which steers each predicted mask toward a spatially connected subgraph, a structural prior with no direct image analogue.

Graph neural networks. The dominant paradigm for graph learning is message passing, with representative MPNNs including GCN (Kipf and Welling, 2017), GIN (Xu et al., 2019), and GatedGCN (Bresson and Laurent, 2017). Graph Transformers (Ying et al., 2021a; Chen et al., 2022) augment local message passing with global self-attention. GraphGPS (Rampášek et al., 2022) runs an MPNN branch and self-attention in parallel per layer. Random Walk Neural Networks (Ivanov and Burnaev, 2018; Tönshoff et al., 2023), in particular NeuralWalker (Chen et al., 2025), encode random-walk sequences with a sequence model, such as a state-space model, and aggregate back to nodes, capturing long-range structural context without the quadratic cost of full attention. Standard MPNNs and Graph Transformers suffer from the symmetry issue, i.e., they assign identical representations to nodes in the same orbit. This ambiguity is problematic for instance-level tasks, where corresponding nodes of different occurrences may receive identical embeddings.

Subgraph matching. Classical algorithms, such as Ullmann's backtracking (Ullmann, 1976) and VF2 (Cordella et al., 2004), compute exact subgraph isomorphisms but run in worst-case exponential time, making them impractical on large graphs. Early neural approaches sidestep combinatorial

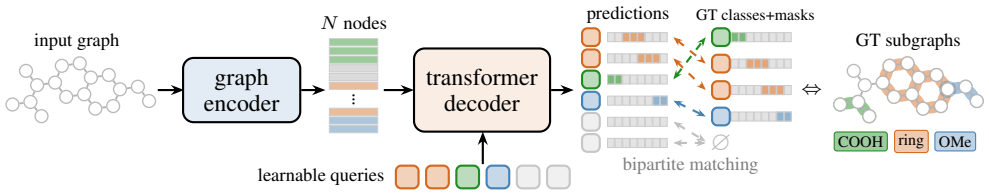


Figure 2: GraphDETR architecture. An input graph is encoded into per-node embeddings by the graph encoder. A set of learnable query vectors and the node embeddings are jointly refined by a (two-way) transformer decoder via interleaved bidirectional cross-attention. A linear class head and a bilinear mask head produce one (class, node-mask) pair per query. At training time, bipartite matching assigns predictions to the ground-truth set of subgraphs.

search by learning graph-level similarity scores. For instance, Graph Matching Networks (Li et al., 2019) employ cross-graph attention to compute pairwise graph similarity, and SimGNN (Bai et al., 2019) combines graph-level embeddings with node-to-node interaction to approximate graph edit distance. These methods measure how similar two graphs are globally, but do not localize where in a larger graph a pattern occurs. Closest to our work is neural subgraph matching (Ying et al., 2021b; Lan et al., 2023; Nguyen et al., 2024), where an arbitrary query graph is given at test time and the model predicts whether and where it occurs in a target graph. These methods condition on one query–target pair at a time and do not jointly predict multiple instances. GraphDETR is the first model to jointly predict all instances of a fixed pattern set in a single differentiable forward pass, with no combinatorial search or post-processing at inference time.

3 GraphDETR for End-to-End Subgraph Detection

The task of subgraph detection requires identifying each occurrence of each pattern separately. The same pattern may appear *multiple* times in one graph, and different occurrences may have *overlapping node sets*. For each occurrence, the model needs to predict which pattern class it belongs to and which nodes it occupies. A suitable model should therefore produce a variable-size set of (class, node-mask) pairs, one per instance.

This prediction problem has a natural analogue in object detection, where a model must produce a variable-size set of (class, localization) pairs, one per detected object. The established paradigm for this task, pioneered by DETR (Carion et al., 2020), is phrased as set prediction, where a fixed set of learnable queries each produce one candidate. GraphDETR adopts this formulation for graphs, replacing bounding boxes with binary node-membership masks and class labels with pattern types.

The model works as follows. A graph neural network encoder produces per-node embeddings that retain the local structure of the input (Section 3.2). A fixed number Q of learnable *query vectors* then interact with these node embeddings through a two-way transformer decoder, with each query predicting a subgraph instance (Section 3.3). A class head and a bilinear mask head decode each query into a (class, node-mask) pair. Queries that do not correspond to any instance predict a designated background class. The model is trained end-to-end with Hungarian matching, which assigns each prediction to a ground-truth instance, and a composite loss that penalizes both misclassification and mask inaccuracy (Section 3.4). An overview of the architecture is visualized in Figure 2.

3.1 Problem Formulation

We formulate subgraph detection as a set prediction task over node-induced subgraph instances. Let $G = (V, E, \boldsymbol{X})$ be a graph with $N = |V|$ nodes and node feature matrix $\boldsymbol{X} \in \mathbb{R}^{N \times d_x}$. A *node-induced subgraph instance* is a pair $(c, \boldsymbol{m})$ consisting of a class label $c \in \{1, \dots, C\}$ and a binary node mask $\boldsymbol{m} \in \{0, 1\}^N$. A mask defines a node subset $V_{\boldsymbol{m}} = \{v \in V : \boldsymbol{m}_v = 1\}$, and the instance is the induced subgraph $G[V_{\boldsymbol{m}}] = (V_{\boldsymbol{m}}, E_{\boldsymbol{m}})$ with $E_{\boldsymbol{m}} = \{(u, v) \in E : u, v \in V_{\boldsymbol{m}}\}$. A graph G may contain multiple instances of each pattern class, forming a set $\mathcal{S}^* = \{(c_k^*, \boldsymbol{m}_k^*)\}_{k=1}^K$, where K varies across graphs and instances of the same or different classes may overlap in their node

sets. We note that this formulation also supports edge attributes (see Section 4) and easily extends to edge-induced subgraphs by replacing node masks with edge-membership masks.

The model is trained on a dataset of annotated graphs $\{(G_i, \mathcal{S}_i^*)\}_{i=1}^n$, where each $\mathcal{S}_i^*$ enumerates the instances present in G_i. At inference time, given an unseen graph G, the goal is to predict a set $\hat{S}$ that matches the complete set S^* of all instances.

3.2 Graph Encoder

The encoder maps each node to an embedding that captures its features and structural context, producing a matrix $\boldsymbol{H} \in \mathbb{R}^{N \times D}$. Any graph neural network that produces per-node embeddings can serve as the backbone. A vital aspect of the encoder is its expressive power, which directly influences the decoder's ability to recover subgraph instances. More precisely, if two nodes receive identical embeddings, the decoder cannot distinguish them and will produce identical mask scores for both across all queries, making it impossible to assign them to different instances. Note that for nodes in the same orbit of the graph's automorphism group, no deterministic equivariant encoder can produce distinct embeddings. However, stochastic mechanisms such as random-walk sampling can break this symmetry, alleviating this issue. Complementary to discriminability, the encoder should produce representations that are structurally informative, encoding sufficient subgraph context for the decoder to determine pattern membership. As an additional source of structural signal, encoders optionally incorporate a Random Walk Structural Encoding (RWSE) (Dwivedi et al., 2022), which enriches each node with random-walk return probabilities at increasing hop distances. We study three encoder families that differ in their expressivity and how they capture structural context.

Message-passing neural networks (MPNNs). We consider GCN (Kipf and Welling, 2017), GIN (Xu et al., 2019), and GatedGCN (Bresson and Laurent, 2017) equipped with the training recipes of Luo et al. (2025). MPNNs iteratively aggregate neighborhood information. Their expressivity is bounded by the 1-WL test (Xu et al., 2019), meaning they assign identical representations to nodes that the 1-WL algorithm fails to distinguish.

Graph transformers. We use GraphGPS (Rampášek et al., 2022), which combines local message passing with global self-attention, allowing each node to aggregate information beyond its immediate neighborhood. This allows each node to aggregate both local structural information and long-range context within each layer. Despite this richer inductive bias, graph transformers remain permutation-symmetric in the standard formulation.

Random-walk neural networks. We employ NeuralWalker (Chen et al., 2025) which combines random walks with message passing. It samples random walks from each node, encodes node and edge features along each walk with identity and adjacency encodings, and processes the resulting sequences with a sequence model. These structural encodings allow the model to identify induced subgraphs along walks, producing node representations that are informed by rich subgraph structure. Moreover, since each walk traces a distinct trajectory, nodes with isomorphic local neighborhoods may receive distinct embeddings, potentially breaking the symmetry that limits MPNNs and standard graph transformers.

3.3 Two-Way Transformer Decoder

The decoder transforms the per-node embeddings from the encoder into Q instance predictions, each consisting of a class label and a node-membership mask. It maintains Q learnable queries $\boldsymbol{Q} \in \mathbb{R}^{Q \times D}$ and node embeddings $\boldsymbol{H} \in \mathbb{R}^{N \times D}$, which are jointly updated by stacking two-way attention blocks, following the bidirectional decoder design of SAM (Kirillov et al., 2023; Ravi et al., 2025). A fixed positional encoding $\boldsymbol{E}_q$ is added to the queries at every layer. We write one block as the following coupled update:

$$\boldsymbol{Q}^{\ell}_{\text{self}} = f_{\text{self}}\left(\boldsymbol{Q}^{\ell} + \boldsymbol{E}_q, \boldsymbol{Q}^{\ell}, \boldsymbol{Q}^{\ell}\right), \quad \boldsymbol{Q}^{\ell}_{\text{cross}} = f_{\text{cross}}\left(\boldsymbol{Q}^{\ell}_{\text{self}} + \boldsymbol{E}_q,\ \boldsymbol{H}^{\ell},\ \boldsymbol{H}^{\ell}\right),$$
$$\boldsymbol{Q}^{\ell+1} = f_{\text{ffn}}\left(\boldsymbol{Q}^{\ell}_{\text{cross}}\right), \qquad \boldsymbol{H}^{\ell+1} = f_{\text{rev}}\left(\boldsymbol{H}^{\ell},\ \boldsymbol{Q}^{\ell} + \boldsymbol{E}_q,\ \boldsymbol{Q}^{\ell}\right),$$

where $f_{\text{self}}, f_{\text{cross}}, f_{\text{rev}}$ denote multi-head attention modules (query–query, query–node, and node–query, respectively), and f_{ffn} is a position-wise feedforward network. Standard LayerNorm and residual connections are applied to each sub-layer.

The update is bidirectional. Queries first interact via self-attention, then gather evidence from nodes, while nodes are updated in parallel by attending back to the queries. The reverse attention induces query-conditioned node representations, allowing the mask head to assign different membership scores to the same node under different queries, which becomes necessary when instances overlap. After the final block, a standard cross-attention ($\boldsymbol{Q} \rightarrow \boldsymbol{H}$) produces $\boldsymbol{Q}_{\text{out}}$ and $\boldsymbol{H}_{\text{ref}}$, which are used by the class and mask heads, respectively.

Class head. A linear projection maps each query to logits over $C + 1$ classes, comprising C foreground pattern classes and a background class $\varnothing$:

$$\hat{\boldsymbol{l}}_j^{\text{cls}} = W_{\text{cls}}\, \boldsymbol{q}_j^{\text{out}} \in \mathbb{R}^{C+1}, \qquad j = 1, \ldots, Q, \tag{1}$$

where $\boldsymbol{q}_j^{\text{out}}$ is the j-th query output from the final decoder block.

Mask head. The mask head predicts node-membership logits through a bilinear interaction between a projected query and the query-conditioned node embeddings:

$$\hat{\boldsymbol{l}}_j^{\text{mask}} = \boldsymbol{H}_{\text{ref}}\, (W_{\text{mask}}\, \boldsymbol{q}_j^{\text{out}})^{\top} \in \mathbb{R}^{N}. \tag{2}$$

Using $\boldsymbol{H}_{\text{ref}}$ rather than the raw encoder output $\boldsymbol{H}$ is critical, because $\boldsymbol{H}_{\text{ref}}$ already encodes which queries are attending to which nodes. Thus, the dot-product score measures how relevant each node is to query j's current hypothesis rather than an unconditional node-query similarity. This gives the mask head access to information that flows in both directions of the decoder.

3.4 Training Objective

The decoder produces a fixed set of Q predictions, while the number of ground-truth instances K varies across graphs, with $K < Q$ in general. Since there is no a priori assignment between predictions and ground-truth instances, training requires a matching step that pairs each ground-truth instance with exactly one prediction. We solve this via the Hungarian algorithm (Kuhn, 1955) and train with a composite loss on the matched pairs.

Hungarian matching. For each graph, we determine an optimal matching $\sigma^* \colon [K] \rightarrow [Q]$ that assigns each ground-truth instance to a distinct query. This is achieved using the Hungarian algorithm applied to a cost matrix $\boldsymbol{C} \in \mathbb{R}^{Q \times K}$ that combines a classification term and mask term:

$$C_{j,k} = \underbrace{-\operatorname{softmax}(\hat{\boldsymbol{l}}_j^{\text{cls}})_{c_k^*}}_{\text{class cost}} + \lambda_{\text{mask}} \underbrace{\mathcal{L}_{\text{BCE}}(\hat{\boldsymbol{l}}_j^{\text{mask}},\, \boldsymbol{m}_k^*)}_{\text{mask cost}} \tag{3}$$

Since the matching is non-differentiable, gradients flow exclusively through the training loss evaluated on the matched pairs. Unmatched queries $j \notin \sigma^*([K])$ are assigned the background class.

Loss function. Given the matching σ^*, the total training loss is given by:

$$\mathcal{L} = \mathcal{L}_{\text{CE}} + \lambda_{\text{mask}}\, \mathcal{L}_{\text{mask}} \tag{4}$$

where $\mathcal{L}_{\text{CE}}$ is a cross-entropy loss over all Q query slots. Unmatched slots target class $\varnothing$ and are down-weighted by $w_{\varnothing} < 1$ to counterbalance the imbalance between the few matched queries and the many background ones. $\mathcal{L}_{\text{mask}}$ is applied only to matched pairs:

$$\mathcal{L}_{\text{mask}} = \mathcal{L}_{\text{BCE}} + \lambda_{\text{cut}}\, \mathcal{L}_{\text{cut}}. \tag{5}$$

The binary cross-entropy term $\mathcal{L}_{\text{BCE}}$ provides per-node supervision.

Graph cut penalty. The term $\mathcal{L}_{\text{cut}}$ encodes a structural prior that subgraph instances are connected subgraphs, penalizing edges that cross the predicted membership boundary. For a matched pair, the penalty is computed from the predicted membership probabilities $p_u = \sigma(l_u)$ as follows:

$$\mathcal{L}_{\text{cut}} = \frac{1}{|\mathcal{E}|} \sum_{(u,v) \in \mathcal{E}} \big[p_u(1 - p_v) + p_v(1 - p_u) \big]. \tag{6}$$

Each summand is zero when both endpoints are confidently on the same side of the boundary ($p \approx 0$ or $p \approx 1$) and reaches 1 when they lie on opposite sides. The gradient drives uncertain boundary nodes toward the majority vote of their neighbors, pushing a node inward if most neighbors are predicted inside the subgraph, and outward otherwise (see Appendix C for a full derivation). This implements a differentiable relaxation of the subgraph connectivity constraint without requiring any auxiliary connected-component check during training.

Table 1: **Functional group detection on ChEMBL12k.** All GraphDETR models use Q=40 queries, 5 encoder layers, D=256, RWSE (K=20), and the full loss. NeuralWalker uses the Mamba sequence backend. Results are mean ± std over 5 seeds.

Encoder	# Params	AP_{100}	$mAP_{50:95}$	Rec_{100}	mIoU	ExactMatch
GCN	3.9M	$0.8669_{\pm 0.0122}$	$0.8766_{\pm 0.0106}$	$0.8631_{\pm 0.0082}$	$0.9768_{\pm 0.0025}$	$0.7424_{\pm 0.0273}$
GIN	4.9M	$0.8815_{\pm 0.0104}$	$0.8888_{\pm 0.0100}$	$0.8724_{\pm 0.0043}$	$0.9792_{\pm 0.0037}$	$0.7642_{\pm 0.0153}$
GatedGCN	5.3M	$0.8687_{\pm 0.0078}$	$0.8751_{\pm 0.0072}$	$0.8735_{\pm 0.0032}$	$0.9756_{\pm 0.0028}$	$0.5948_{\pm 0.0259}$
GraphGPS	6.6M	$0.9191_{\pm 0.0112}$	$0.9230_{\pm 0.0099}$	$0.8923_{\pm 0.0017}$	$0.9812_{\pm 0.0012}$	$0.7460_{\pm 0.0286}$
NeuralWalker (λ_{mask}=2)	7.4M	$0.9136_{\pm 0.0165}$	$0.9207_{\pm 0.0164}$	$0.9567_{\pm 0.0102}$	$0.9912_{\pm 0.0036}$	$\mathbf{0.8912}_{\pm 0.0264}$
NeuralWalker (λ_{mask}=4)	7.4M	$\mathbf{0.9370}_{\pm 0.0147}$	$\mathbf{0.9438}_{\pm 0.0135}$	$\mathbf{0.9718}_{\pm 0.0037}$	$\mathbf{0.9944}_{\pm 0.0009}$	$0.8342_{\pm 0.0320}$

4 Molecular Functional Group Detection

We apply GraphDETR to molecular functional group (FG) detection, an instance-level subgraph detection problem where both the class and the exact atom-membership mask of every FG instance need to be jointly recovered. We evaluate GraphDETR on ChEMBL[1], drawn from ChEMBL-36, containing 2,827,875 molecules across 95 functional group classes. Notably, functional group labels are context-sensitive, as an instance's class can depend on atoms outside its mask rather than solely on the matched subgraph. For ablation studies we use ChEMBL12k, a subset of 12,000 molecules. Full dataset details can be found in Appendix E.

Evaluation metrics. We adopt an object detection evaluation protocol adapted to graphs. Each predicted node-set mask is matched to a ground-truth instance by Hungarian assignment, using the Intersection over Union (IoU) of their node sets, i.e., $|V_{\text{pred}} \cap V_{\text{gt}}| \,/\, |V_{\text{pred}} \cup V_{\text{gt}}|$, as the similarity measure. We report five metrics: $\mathbf{AP}_{100}$ (average precision at IoU $= 1.0$, macro-averaged over classes), **mAP** (COCO-style, thresholds 0.50–0.95 in steps of 0.05), $\mathbf{Rec}_{100}$ (recall at IoU $= 1.0$), **mIoU** (mean mask IoU over matched true-positive pairs), and **ExactMatch** (fraction of molecules where every GT instance is matched with IoU ≥ 0.5 and no false positive is active).

Baselines. As no existing method directly addresses multi-instance subgraph detection, we adapt NeuroMatch (Ying et al., 2021b) as a neural baseline (Appendix D). Since it only tests whether a query occurs around a given node, it fails to recover instance masks (AP_{100} below 1% on ChEMBL12k, Cactus40, and Cliques100) while being over two orders of magnitude slower than GraphDETR. Runtime comparisons against the exact solver VF2 (Cordella et al., 2004) are reported in Section 5.

4.1 Graph Encoder Comparison

We start by comparing the encoder backbones introduced in Section 3.2, keeping all other components fixed. As shown in Table 1, GraphDETR achieves strong detection performance across all encoders, demonstrating that the set prediction formulation is effective for this combinatorially rich real-world task. Notably, local MPNN encoders (GCN, GIN, GatedGCN) are consistently outperformed by GraphGPS and NeuralWalker across all precision-oriented metrics, with NeuralWalker achieving the best overall performance. This likely stems from the stronger ability of GraphGPS and NeuralWalker to capture global structural context, in contrast to the purely local aggregation of MPNNs.

Among local encoders, GatedGCN achieves competitive AP_{100} but its ExactMatch (i.e., the ability to recover all instances of a molecule without false positives) is substantially lower than both GCN and GIN, suggesting that it struggles to cleanly separate co-occurring instances at the molecule level. Furthermore, mIoU is uniformly high across all encoders, which shows that matched pairs are well localized regardless of backbone. Differences between encoders are therefore most pronounced in recall and ExactMatch rather than in the quality of masks for already-identified instances.

Data scaling (Figure 3). We further evaluate how detection performance scales with training data size on the full ChEMBL dataset. Both GIN and NeuralWalker improve log-linearly as the number of training molecules grows, with NeuralWalker consistently ahead. The parallel slopes suggest that NeuralWalker's advantage over GIN does not erode with scale, pointing to a structural rather than a data-driven benefit.

[1] https://chembl.gitbook.io/

Table 2: **Loss ablation (OFAT)** on the ChEMBL12k dataset. Each row adds one loss component relative to the previous.

Loss	AP_{100}	ExactMatch	Rec_{100}
CE only	0.4	0.8	0.5
+ BCE	85.4	79.1	95.0
+ BG weight	92.4	84.8	97.1
Full (+ cut)	94.3	81.8	97.2

Figure 3: **Data scaling** on ChEMBL (AP_{100}, test set). Encoders improve log-linearly with scale, with NeuralWalker consistently ahead.

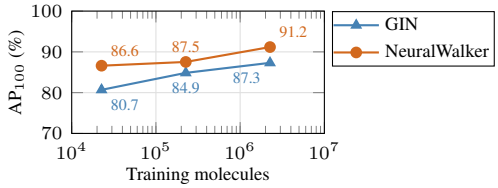


Table 3: **Decoder ablation (OFAT)** on the ChEMBL12k validation set. Baseline: NeuralWalker encoder with # layers=3, # queries=40, h=8 heads, attn downsample rate r=2. Each row varies exactly one axis.

Varied axis	**Value**	**AP_{100}**	**mAP**	**Rec_{100}**	**mIoU**	**ExactMatch**
baseline	–	**0.9429**	**0.9497**	0.9720	0.9939	0.8180
# layers	1	0.8983	0.9050	0.9493	0.9885	0.8100
# layers	2	0.9207	0.9273	0.9690	0.9936	0.8460
# queries	10	0.8674	0.8822	0.9107	0.9858	0.7080
# queries	20	0.9296	0.9371	0.9563	0.9924	0.8180
# queries	30	0.9166	0.9247	0.9635	0.9925	0.8420
# attn heads	4	0.8953	0.9064	0.9502	0.9893	0.7020
attn downsample rate	1	0.9062	0.9169	0.9639	0.9923	0.8500
attn downsample rate	4	0.9312	0.9411	**0.9739**	**0.9949**	**0.9000**

4.2 Ablation and Hyperparameter Studies

In this section, we ablate key components of GraphDETR and study the sensitivity to central hyperparameters on the ChEMBL12k dataset. Specifically, we examine the contribution of each loss term, the impact of the decoder capacity, and the sensitivity of the NeuralWalker encoder to its hyperparameter choices. Qualitative predictions on held-out test molecules are provided in Appendix G.

Loss components (Table 2). We ablate the training objective introduced in Section 3.4 by progressively adding each loss term. Without mask supervision, the model fails to localize instances entirely, confirming that cross-entropy supervision (CE) on class predictions alone is insufficient for instance-level detection. Adding the binary cross-entropy mask loss (BCE) recovers most of the performance, and down-weighting the background class (BCE+BG weight) further reduces false positive predictions substantially. Finally, the full objective, which additionally includes the graph cut penalty (Full+cut), further improves AP_{100} and Rec_{100}, though at a slight cost to ExactMatch.

Decoder capacity (Table 3). We study the effect of the main architectural hyperparameters of the two-way transformer decoder (Section 3.3), *i.e.,* the number of attention blocks, query slots, and attention heads. Furthermore, we analyze the effect of the downsample rate r, which compresses the attention key/value dimensions to D/r. Reducing the number of attention blocks degrades AP_{100} gradually, suggesting that decoder depth contributes meaningfully to detection performance. Concerning the number of query slots Q, it can be observed that Q=10 leads to a sharp performance drop, possibly because many molecules contain more than 10 functional group instances, leaving the decoder unable to cover them all simultaneously. The rather gradual degradation observed at higher values of Q may reflect the decreasing fraction of molecules affected as Q grows. Turning to the number of attention heads, we find that halving from 8 to 4 causes a larger performance drop than any reduction in depth, showing that multi-head diversity is particularly important for decoder performance. Finally, increasing the attention downsample rate r improves ExactMatch notably at a mild cost to AP_{100}, possibly suggesting that compressing the key/value space acts as a form of regularization that promotes more decisive mask predictions.

NeuralWalker encoder sensitivity (Table 4). Given that NeuralWalker achieves the best overall performance in Section 4.1, we study the sensitivity of its key parameter choices, i.e., the number of NeuralWalker blocks, walk length, and the use of RWSE and edge features. The number of

Table 4: **NeuralWalker encoder ablation (OFAT)** on the ChEMBL12k validation set.

Varied axis	Value	Layers L	Walk ℓ	RWSE	AP_{100}	mAP	Rec_{100}	mIoU	ExactMatch
baseline	–	5	20	✓(K=20)	**0.9429**	**0.9497**	**0.9720**	**0.9939**	0.8180
# layers	2	2	20	✓	0.9122	0.9230	0.9617	0.9910	0.8380
# layers	3	3	20	✓	0.9130	0.9219	0.9688	0.9930	**0.8530**
# layers	4	4	20	✓	0.9352	0.9428	0.9715	0.9931	0.8480
walk length	10	5	10	✓	0.9292	0.9344	0.9581	0.9918	0.8500
walk length	40	5	40	✓	0.8931	0.9021	0.9625	0.9931	0.8540
RWSE	✗	5	20	✗	0.9225	0.9292	0.9635	0.9921	0.8480
edge attr	off	5	20	✓	0.9170	0.9246	0.9678	0.9921	0.8240

NeuralWalker blocks L has only a modest effect on performance across the tested values, with AP_{100} improving gradually with depth while ExactMatch peaks at an intermediate value and then slightly declines. Turning to walk length, an optimum is observed at ℓ=20, with shorter walks providing insufficient structural context and, interestingly, longer walks actively hurting performance. Removing RWSE, which enriches node features with random-walk return probabilities, leads to a meaningful drop in AP_{100}, suggesting that it provides further structural information beyond what the walk sequences themselves capture. Finally, ablating edge features also leads to a meaningful drop in AP_{100}, confirming that edge attributes provide further useful structural signal for instance detection.

5 General Subgraph Detection Capabilities

To assess the generality of GraphDETR beyond molecular graphs, we evaluate it on a diverse set of subgraph detection benchmarks covering exact and approximate pattern matching across a wide range of graph sizes and pattern types. In addition, we provide inference runtimes and study the ability of trained models to generalize to larger graphs than those seen during training.

We evaluate on five dataset families: Cactus graphs in which cycles of varying lengths are the query patterns, bipartite graphs with injected k-cliques as query patterns, ZINC12k (Dwivedi et al., 2023) where frequent molecular subgraphs serve as query patterns, Mol-Reddit where molecular structures injected into social network host graphs are the targets of detection, and fuzzy variants of the Cactus and Clique datasets where instances only approximately match the query pattern. For the Cactus and Clique datasets, the graph generation procedure is designed to prevent accidental pattern occurrences, ensuring complete ground-truth annotations. For ZINC12k, completeness is guaranteed by the exhaustive frequent subgraph enumeration. Full details are provided in Appendix E. We report AP_{100} as the primary metric, following the same evaluation protocol as in Section 4. Model configuration follows Section 4, with hyperparameters listed in Appendix F.2.

Exact detection (Table 5). We start by examining exact subgraph detection across the synthetic benchmarks. As in Section 4.1, NeuralWalker and GraphGPS consistently outperform local MPNN encoders across all datasets, confirming that global structural context is beneficial beyond the molecular setting. Performance on Cactus graphs degrades substantially with graph size, which is expected given that both the host graphs and the cycle patterns grow larger, making disambiguation increasingly difficult. In contrast, performance on Clique graphs remains strong even as the clique size grows, suggesting that the dense connectivity of cliques provides a distinctive structural signature that is easier to detect regardless of size. ZINC12k and Mol-Reddit present a different challenge as they involve attributed graphs with molecular (sub-)structures as patterns. Notably, NeuralWalker performs particularly strongly on Mol-Reddit, which is arguably the most challenging dataset due to its incomplete annotations and large, varied host graphs.

Approximate detection (Table 5). We further evaluate GraphDETR on approximate pattern matching, where instances are defined by their proximity to a canonical query pattern under graph edit distance (GED). The task requires separating instances whose GED to the canonical pattern falls below a threshold from those that do not. GraphDETR achieves surprisingly strong performance on both fuzzy datasets, demonstrating that it can learn to implicitly discriminate between structurally similar subgraphs purely from data, without any explicit GED computation. This is particularly striking as the model must correctly reject subgraphs that differ from the query pattern by only a small number of edits, a fine-grained structural distinction that emerges purely from training.

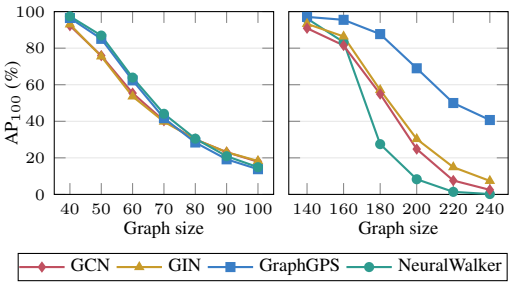
Table 5: **Synthetic subgraph detection** in AP_{100}%. n/m denotes the average number of nodes/edges.

Dataset	Patterns (#)	Attr.	Stats	GraphDETR encoder			
				GCN	GIN	GraphGPS	NeuralWalker
Cactus40	(3-8)-cycles (6)	✗	$n = 40, m = 45$	$92.33_{\pm 0.56}$	$92.98_{\pm 0.90}$	$96.51_{\pm 0.43}$	$\mathbf{97.44}_{\pm 0.44}$
Cactus200	(11-16)-cycles (6)	✗	$n = 200, m = 204$	$42.80_{\pm 3.16}$	$41.51_{\pm 4.45}$	$83.67_{\pm 0.78}$	$\mathbf{91.90}_{\pm 2.41}$
Cactus1000	(27-32)-cycles (6)	✗	$n = 1000, m = 1004$	$10.46_{\pm 3.04}$	$4.82_{\pm 7.30}$	$40.22_{\pm 0.68}$	$\mathbf{42.20}_{\pm 2.06}$
Cliques100	5-clique (1)	✗	$n = 140, m = 350$	$90.94_{\pm 2.48}$	$93.31_{\pm 0.74}$	$\mathbf{97.05}_{\pm 1.60}$	$96.15_{\pm 0.57}$
Cliques200	10-clique (1)	✗	$n = 290, m = 1450$	$90.33_{\pm 1.46}$	$92.84_{\pm 2.20}$	$\mathbf{96.45}_{\pm 1.17}$	$93.48_{\pm 2.35}$
Cliques400	20-clique (1)	✗	$n = 590, m = 5900$	$92.51_{\pm 2.85}$	$94.12_{\pm 1.51}$	$\mathbf{95.05}_{\pm 3.23}$	$92.58_{\pm 3.48}$
ZINC12k	frequent subgraphs (9)	✓	$n = 23, m = 25$	$89.68_{\pm 0.88}$	$92.86_{\pm 0.25}$	$95.28_{\pm 0.13}$	$\mathbf{97.66}_{\pm 0.27}$
Mol-Reddit	injected molecules (10)	✗	$n = 441, m = 517$	$65.68_{\pm 1.11}$	$62.12_{\pm 1.31}$	$80.78_{\pm 1.32}$	$\mathbf{89.76}_{\pm 0.97}$
Cactus-fuzzy	fuzzy (7-10)-cycles (4)	✓	$n = 100, m = 105$	$67.78_{\pm 1.17}$	$76.19_{\pm 2.03}$	$84.86_{\pm 0.51}$	$\mathbf{90.91}_{\pm 2.88}$
Cliques-fuzzy	fuzzy 8-clique (1)	✗	$n = 170, m = 470$	$95.05_{\pm 1.02}$	$95.26_{\pm 0.89}$	$\mathbf{96.66}_{\pm 0.86}$	$93.94_{\pm 2.10}$

Table 6: **Inference runtime (s / graph).** VF2 uses a 60 s per-graph budget; † marks datasets where VF2 timed out on >80% of test graphs. GraphDETR runtimes are means across seeds.

Dataset		GraphDETR			
	VF2	GCN	GIN	GPS	NW
Cactus40	0.068	5.4e-5	5.2e-5	9.8e-5	0.0014
Cactus200	2.799	9.3e-5	9.2e-5	0.0002	0.0174
Cactus1000	60†	0.0002	0.0002	0.0009	0.0412
Cliques100	0.120	5.8e-5	6.2e-5	0.0001	0.0041
Cliques200	60†	8.5e-5	9.0e-5	0.0003	0.0038
Cliques400	60†	0.0002	0.0002	0.0007	0.0079
ZINC12k	0.010	5.5e-5	5.3e-5	9.9e-5	0.0021
Mol-Reddit	51†	0.0002	0.0002	0.0021	0.0125
Cactus-fuzzy	–	5.6e-5	5.6e-5	0.0001	0.0045
Cliques-fuzzy	–	6.1e-5	6.6e-5	0.0002	0.0043

Figure 4: **Generalization to larger graphs.** AP_{100} of a single model trained on small graphs, evaluated at increasing graph sizes. Left: trained on Cactus40. Right: trained on Cliques100.



Runtime (Table 6). We compare the inference runtime of GraphDETR against VF2 (Cordella et al., 2004), which is the de facto standard algorithm for subgraph isomorphism, a highly optimized combinatorial baseline. VF2 fails to terminate within the provided time budget on a large amount of test graphs across several datasets, while GraphDETR processes all graphs in a single forward pass at runtimes several orders of magnitude lower. This gap widens with graph size, consistent with the exponential worst-case complexity of combinatorial subgraph isomorphism search, while GraphDETR maintains low runtimes even on the largest graphs.

Generalization to larger graphs (Figure 4). We evaluate the ability of a single model trained on small graphs (Cactus40 and Cliques100) to generalize to increasingly larger graphs at test time. For cycles, all encoders significantly degrade as graph size grows, with NeuralWalker and GraphGPS degrading slightly slower than MPNNs. For cliques, the picture is more nuanced and somewhat surprising. GraphGPS proves to be the most robust encoder, maintaining strong performance well beyond the training distribution, while NeuralWalker degrades more significantly at larger graph sizes despite being the best encoder in-distribution.

6 Conclusion

This work introduced GraphDETR, an end-to-end framework for subgraph detection that casts the problem as set prediction. By combining graph encoders with a two-way transformer decoder and bipartite matching, the model predicts the complete set of subgraph instances in a single forward pass, eliminating the need for combinatorial search or post-processing. This formulation supports both exact and approximate matching, extending beyond the rigid constraints of classical subgraph isomorphism. Empirical results across molecular and synthetic benchmarks demonstrate that GraphDETR achieves strong detection performance, scales to large graphs, and operates with orders-of-magnitude faster inference than traditional algorithms. The analysis further highlights the importance of expressive graph encoders and the role of global structural context in instance-level graph reasoning. More broadly, the proposed set prediction perspective provides a general and flexible framework for structured prediction on graphs and a useful framework for evaluating expressive graph encoders. Future work may explore richer query conditioning, improved scalability to even larger graphs, and extensions to other structured outputs such as edge-induced subgraphs or hierarchical patterns.

REPRODUCIBILITY STATEMENT

The model architecture and training objective are described in Section 3. Dataset construction, including the synthetic data generation procedures, is detailed in Appendix E, and all hyperparameters, search grids, and computing resources are listed in Appendix F. The NeuroMatch baseline adaptation is described in Appendix D. The main results are reported as mean and standard deviation over 5 random seeds. Code and generated datasets will be made publicly available upon acceptance.

AI USE STATEMENT

In this work, we used generative AI tools to partially assist in implementing methods and generating synthetic data sets. Specifically, LLMs helped us write code to integrate existing GNN codebases into our code and to generate our synthetic datasets, while we designed the methods and the data generation procedure. We also used the LLM-based feedback tool provided by ICLR on a draft of this paper. Based on its feedback, we revised formulations on the graph cut penalty, which we verified by hand, and clarified some further statements in the text. It further included feedback on our experiments, which we did not act upon. We have not used generative AI tools to help develop theoretical models or conceptual frameworks, provide critical ingredients for proving mathematical claims, assist in the writing of proofs, propose or refine hypotheses, assist with translation, clean and reformat datasets, support qualitative and thematic data analysis, or interpret results. Additionally, we used generative AI tools to create or edit code, specifically for plotting figures, job-launching scripts, and code documentation, and to edit the paper to improve readability by rephrasing individual sentences.

We have reviewed all AI-assisted work, and all LLM-generated or modified code was manually checked. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Yunsheng Bai, Hao Ding, Song Bian, Ting Chen, Yizhou Sun, and Wei Wang. SimGNN: A neural network approach to fast graph similarity computation. In *ACM International Conference on Web Search and Data Mining (WSDM)*, pages 384–392, 2019.

Guy W Bemis and Mark A Murcko. The properties of known drugs. 1. molecular frameworks. *Journal of medicinal chemistry*, 39(15):2887–2893, 1996.

Xavier Bresson and Thomas Laurent. Residual gated graph convnets. *arXiv preprint arXiv:1711.07553*, 2017.

Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 213–229. Springer, 2020.

Dexiong Chen, Leslie O'Bray, and Karsten Borgwardt. Structure-aware transformer for graph representation learning. In *International Conference on Machine Learning (ICML)*, pages 3469–3489. PMLR, 2022.

Dexiong Chen, Till Hendrik Schulz, and Karsten Borgwardt. Learning long range dependencies on graphs via random walks. In *International Conference on Learning Representations (ICLR)*, 2025. URL https://openreview.net/forum?id=kJ5H7oGT2M.

Bowen Cheng, Ishan Misra, Alexander G Schwing, Alexander Kirillov, and Rohit Girdhar. Masked-attention mask transformer for universal image segmentation. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1290–1299, 2022.

Gonzalo Colmenarejo. Efgs: a complete and accurate implementation of Ertl's functional group detection algorithm in RDKit. *Journal of Chemical Information and Modeling*, 65(3):1061–1066, 2025.

Luigi P Cordella, Pasquale Foggia, Carlo Sansone, and Mario Vento. A (sub) graph isomorphism algorithm for matching large graphs. *Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 26(10):1367–1372, 2004.

Vijay Prakash Dwivedi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Graph neural networks with learnable structural and positional representations. In *International Conference on Learning Representations (ICLR)*, 2022.

Vijay Prakash Dwivedi, Chaitanya K Joshi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks. *Journal of Machine Learning Research (JMLR)*, 24(43):1–48, 2023.

Peter Ertl. An algorithm to identify functional groups in organic molecules. *Journal of Cheminformatics*, 9(1):36, 2017. doi: 10.1186/s13321-017-0225-z.

Dedre Gentner. Structure-mapping: A theoretical framework for analogy. *Cognitive science*, 7(2): 155–170, 1983.

Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, 2020.

Sergey Ivanov and Evgeny Burnaev. Anonymous walk embeddings. In *International Conference on Machine Learning (ICML)*, pages 2186–2195. PMLR, 2018.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2017. URL https://openreview.net/forum?id=SJU4ayYgl.

Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C Berg, Wan-Yen Lo, et al. Segment anything. In *Proceedings of the International Conference on Computer Vision (ICCV)*, pages 4015–4026, 2023.

Harold W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97, 1955.

Zixun Lan, Limin Yu, Linglong Yuan, Zili Wu, Qiang Niu, and Fei Ma. Sub-gmn: The neural subgraph matching network model. In *2023 16th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*, pages 1–7. IEEE, 2023.

Yujia Li, Chenjie Gu, Thomas Dullien, Oriol Vinyals, and Pushmeet Kohli. Graph matching networks for learning the similarity of graph structured objects. In *International Conference on Machine Learning (ICML)*, pages 3835–3845. PMLR, 2019.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations (ICLR)*, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

Yuankai Luo, Lei Shi, and Xiao-Ming Wu. Can classic gnns be strong baselines for graph-level tasks? simple architectures meet excellence. In *International Conference on Machine Learning (ICML)*, pages 41290–41310. PMLR, 2025.

Duc Q Nguyen, Thanh Toan Nguyen, Jun Jo, Florent Poux, Shikha Anirban, and Tho T Quan. Explainable neural subgraph matching with learnable multi-hop attention. *IEEE access*, 12: 130474–130492, 2024.

Siegfried Nijssen and Joost N Kok. A quickstart in frequent structure mining can make a difference. In *Proceedings of the international conference on Knowledge discovery and data mining (KDD)*, pages 647–652, 2004.

Eric Plotnick. *Concept mapping: A graphical system for understanding the relationship between concepts*. ERIC Clearinghouse on Information and Technology Syracuse, NY, 1997.

Ladislav Rampášek, Michael Galkin, Vijay Prakash Dwivedi, Anh Tuan Luu, Guy Wolf, and Dominique Beaini. Recipe for a general, powerful, scalable graph transformer. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, 2022.

Nikhila Ravi, Valentin Gabeur, Yuan-Ting Hu, Ronghang Hu, Chaitanya Ryali, Tengyu Ma, Haitham Khedr, Roman Rädle, Chloe Rolland, Laura Gustafson, et al. SAM 2: Segment anything in images and videos. In *International Conference on Learning Representations (ICLR)*, 2025.

John W Raymond, Eleanor J Gardiner, and Peter Willett. Heuristics for similarity searching of chemical graphs using a maximum common edge subgraph algorithm. *Journal of Chemical Information and Computer Sciences*, 42(2):305–316, 2002.

Jan Tönshoff, Martin Ritzert, Hinrikus Wolf, and Martin Grohe. Walking out of the weisfeiler leman hierarchy: Graph learning beyond message passing. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL https://openreview.net/forum?id=vgXnEyeWVY.

Julian R Ullmann. An algorithm for subgraph isomorphism. *Journal of the ACM (JACM)*, 23(1): 31–42, 1976.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations (ICLR)*, 2019.

Pinar Yanardag and Svn VN Vishwanathan. Deep graph kernels. In *International conference on knowledge discovery and data mining (KDD)*, pages 1365–1374, 2015.

Qingwu Yang and Sing-Hoi Sze. Path matching and graph matching in biological networks. *Journal of Computational Biology*, 14(1):56–67, 2007.

Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. Do transformers really perform badly for graph representation? In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 34, pages 28877–28888, 2021a.

Zhitao Ying, Andrew Wang, Jiaxuan You, Chengtao Wen, Arquimedes Canedo, and Jure Leskovec. Neural subgraph matching, 2021b. URL https://openreview.net/forum?id=LMslR3CTzE_.

Barbara Zdrazil, Eloy Felix, Fiona Hunter, Emma J Manners, James Blackshaw, Sybilla Corbett, Marleen De Veij, Harris Ioannidis, David Mendez Lopez, Juan F Mosquera, et al. The chembl database in 2023: a drug discovery platform spanning multiple bioactivity data types and time periods. *Nucleic acids research*, 52(D1):D1180–D1192, 2024.

Hao Zhang, Feng Li, Shilong Liu, Lei Zhang, Hang Su, Jun Zhu, Lionel Ni, and Heung-Yeung Shum. DINO: DETR with improved denoising anchor boxes for end-to-end object detection. In *International Conference on Learning Representations (ICLR)*, 2022.

Appendix

The appendix provides supplementary technical and experimental materials. It is organized as follows: Section A discusses the limitations of the proposed approach. Section B provides a discussion on the broader impacts. Section C provides the derivation and analysis of the graph cut penalty used in the training objective. Section D describes the NeuroMatch baseline. Section E provides details of the datasets used in this work. Section F provides full implementations for reproducibility purposes. Section G provides additional experimental results.

A Limitations

While GraphDETR demonstrates strong empirical performance and scalability, several limitations remain.

First, the approach relies on a fixed number of query slots, which imposes an upper bound on the number of detectable instances per graph. Although this can be mitigated by choosing a sufficiently large number of queries, doing so increases computational cost and may introduce optimization challenges due to a larger proportion of unmatched queries.

Second, the model's performance depends critically on the expressivity of the graph encoder. As discussed in Section 3.2, encoders with limited discriminative power (*e.g.,* standard MPNNs) may fail to distinguish structurally symmetric nodes, which can hinder instance separation. While more expressive architectures, such as Graph Transformers or random-walk-based models, alleviate this issue, they may incur higher computational overhead, and no single model dominates on all datasets.

Third, although GraphDETR scales well compared to combinatorial methods, its memory and runtime still grow with graph size due to attention operations in the decoder. This may limit applicability to extremely large graphs without further architectural modifications, such as sparse or hierarchical attention.

Fourth, the current formulation is limited to a closed vocabulary of pattern classes observed during training. The model cannot directly generalize to detect entirely novel or unseen subgraph patterns at inference time, in contrast to open-vocabulary detection settings explored in other domains. Enabling open-vocabulary or query-conditioned subgraph detection, where patterns are specified at test time, remains an important direction for future work.

Finally, the current formulation assumes access to fully annotated training data with instance-level supervision (i.e., node masks). In many real-world settings, such annotations are costly or unavailable. Extending the framework to weakly supervised or unsupervised settings remains an open challenge.

B Broader Impacts

GraphDETR provides a general framework for detecting structured patterns in graphs, with potential applications across chemistry, biology, and network science.

In molecular science, learned substructure detectors could be integrated into differentiable pipelines for tasks such as property prediction, reaction prediction, or retrosynthesis planning.

In network analysis, the method may facilitate the discovery of recurring structural motifs in social, biological, or technological networks, contributing to a deeper understanding of their organization and function.

However, these capabilities also raise potential concerns. In chemistry and biology, improved substructure detection could be misused to design harmful compounds or bypass safety constraints in molecular screening pipelines. In social network analysis, identifying structural patterns at scale may raise privacy concerns if applied to sensitive relational data.

Mitigating these risks requires responsible deployment, including adherence to domain-specific regulations, careful dataset curation, and consideration of ethical implications. We emphasize that GraphDETR is a general-purpose modeling framework, and its societal impact depends on the context in which it is applied.

C Graph Cut Penalty: Derivation and Gradient Analysis

Here we describe in more detail the graph cut penalty introduced in Section 3.4.

C.1 Loss definition

For a matched pair with predicted membership probabilities $p_u = \sigma(l_u)$ (where σ is the sigmoid function and l_u is the per-node logit), the graph cut penalty is

$$\mathcal{L}_{\text{cut}} = \frac{1}{|\mathcal{E}|} \sum_{(u,v)\in\mathcal{E}} \big[p_u(1-p_v) + p_v(1-p_u)\big]. \tag{7}$$

Each summand is the product $p_u(1-p_v)$ (node u inside, node v outside) plus $p_v(1-p_u)$ (the reverse), and equals zero when both endpoints are on the same side of the boundary ($p \approx 0$ or $p \approx 1$) and reaches its maximum of 1 when they straddle it ($p_u = 1, p_v = 0$).

C.2 Gradient analysis

Differentiating equation 7 with respect to logit l_u, and using $\frac{\partial p_u}{\partial l_u} = p_u(1-p_u)$:

$$\frac{\partial \mathcal{L}_{\text{cut}}}{\partial l_u} = \frac{p_u(1-p_u)}{|\mathcal{E}|} \sum_{v\,:\,(u,v)\in\mathcal{E}} (1-2p_v). \tag{8}$$

Three structural properties of this gradient are noteworthy:

- **Saturated nodes are unaffected.** The pre-factor $p_u(1-p_u)$ is zero when $p_u \in \{0,1\}$, so nodes that are already certain do not receive any gradient from this term, avoiding perturbation of confident predictions.
- **Majority-vote pressure.** The sum $\sum_v (1-2p_v)$ is negative when the majority of neighbors have $p_v > 1/2$ (i.e. are predicted inside the subgraph), and positive otherwise. A negative sum decreases the gradient, increasing l_u, hence pushing u further inside; a positive sum pushes u outward. The penalty therefore implements a differentiable neighbor-majority constraint.
- **Boundary sharpening.** The gradient is largest for boundary nodes ($p_u \approx 1/2$) whose neighbors agree, and vanishes at topological interiors or exteriors, focusing learning signal exactly where structural ambiguity is highest.

D NeuroMatch Baseline

NeuroMatch (Ying et al., 2021b) performs anchored pairwise containment testing. Given a query graph, a target neighborhood, and an anchor node in each, it predicts whether the rooted query is contained in the rooted target. It does not natively enumerate multiple occurrences, assign pattern classes, recover node masks, or suppress duplicate detections. To evaluate it on our benchmarks, we therefore build a detection pipeline around the original implementation.

Adaptation. For training, we select a canonical query exemplar for each pattern class, construct supervised positive and negative query–target pairs, and extend the model with node and edge attributes, which the original implementation does not support. At inference, we exhaustively evaluate all anchor–class pairs, accept candidates using calibrated thresholds, recover node masks from accepted anchors via region growing, suppress duplicate detections, and convert the predictions into the set-based output format of GraphDETR. None of these components are part of NeuroMatch, and they introduce design choices that affect the final result.

Modification details. We encountered several issues during adaptation. First, the non-negativity activation in the order-embedding model occasionally led to embedding collapse. Second, the original unweighted calibration classifier collapsed to the majority class under the imbalanced pair distribution, which we mitigated with class weighting. Third, the pair distribution seen during training differs substantially from the exhaustive candidate distribution at detection time, causing over- or under-detection depending on the threshold.

Results. Table 7 reports the results. The adapted NeuroMatch fails to recover instance masks on all three datasets and is over two orders of magnitude slower, since inference requires evaluating many anchor–class combinations rather than predicting all instances in one pass.

Table 7: Adapted NeuroMatch vs. GraphDETR. Runtime in seconds per graph.

Dataset	**Method**	**AP$_{100}$**	**mAP**	**Rec$_{100}$**	**ExactMatch**	**Runtime**
ChEMBL12k	NeuroMatch	0.0	0.0	0.27	0.0	0.5938
	GraphDETR	93.70	94.38	97.18	83.42	0.0002
Cactus40	NeuroMatch	0.30	0.88	22.68	0.0	0.3016
	GraphDETR	97.44	98.27	98.66	84.02	0.0014
Cliques100	NeuroMatch	0.0	0.0	0.24	0.0	0.5008
	GraphDETR	97.05	98.16	98.02	96.00	0.0001

E Dataset Descriptions

We provide details of the creation of all datasets used in this work.

E.1 Molecular datasets

Table 8: **Statistics of the ChEMBL and ChEMBL12k dataset.**

Dataset	# Graphs	# Classes	Patterns/Graph	Nodes ($\mu \pm \sigma$)	Edges ($\mu \pm \sigma$)	Nodes [min,max]	Edges [min,max]
ChEMBL	2,827,875	95	5.8 ± 2.9	29.8 ± 10.4	32.4 ± 11.4	[1.0,100.0]	[0.0,132.0]
ChEMBL12k	12,000	95	5.8 ± 2.9	29.9 ± 10.6	32.5 ± 11.5	[5.0,100.0]	[4.0,108.0]

ChEMBL. We construct the ChEMBL dataset from ChEMBL-36 (Zdrazil et al., 2024) using a scalable preprocessing pipeline. Starting from 2,848,825 molecules, we retain those parseable by RDKit and remove duplicates, yielding 2,827,875 molecules. Functional group instances and their atom-level membership masks are generated by Ertl's IFG algorithm (Ertl, 2017; Colmenarejo, 2025), which provides context-sensitive labeling (*e.g.,* distinguishing amide nitrogen from a free amine). After filtering functional group classes with frequency below 0.1%, the vocabulary contains 95 classes. Molecules are featurized with OGB atom and bond features (Hu et al., 2020) (9-dimensional atom features, 3-dimensional bond features). The dataset is split 80/10/10 by Bemis–Murcko scaffold (Bemis and Murcko, 1996) to ensure no training scaffold appears at test time.

ChEMBL12k. For ablation studies and rapid iteration, we define a 12,000-molecule subset drawn uniformly at random from ChEMBL: 10,000 molecules for training, 1,000 for validation, 1,000 for test. The subset shares the same atom/bond features and FG vocabulary as the full ChEMBL dataset; the split is fixed by a global seed for reproducibility. The statistics of both datasets are provided in Table 8.

E.2 Synthetic datasets

All synthetic datasets are stored in Gaston format, with a target-graph file and a pattern file. Each sample is a host graph together with a set of injected subgraph instances annotated by class label and node mask. Statistics for each dataset are listed in Table 9. Query patterns for each dataset are described in Table 10.

Table 9: **Statistics of all synthetic datasets.**

Dataset	# Graphs	# Classes	Patterns/Graph	Nodes ($\mu\pm\sigma$)	Edges ($\mu\pm\sigma$)	Nodes [min,max]	Edges [min,max]
Cactus40	12000	6	6.3±1.2	40.0±0.0	45.3±1.2	[40,40]	[41,49]
Cactus200	12000	6	4.8±0.9	200.0±0.0	203.8±0.9	[200,200]	[201,207]
Cactus1000	12000	6	4.9±0.9	1000.0±0.0	1003.9±0.9	[1000,1000]	[1001,1007]
Cliques100	12000	1	10.0±0.0	139.5±0.7	350.3±15.0	[135,140]	[298,412]
Cliques200	12000	1	10.0±0.0	290.0±0.1	1450.3±30.0	[288,290]	[1346,1564]
Cliques400	12000	1	10.0±0.0	590.0±0.0	5900.2±60.2	[590,590]	[5643,6186]
ZINC12k	12000	9	5.5±3.3	23.2±4.5	24.9±5.3	[9,37]	[8,42]
Mol-Reddit	11929	10	2.1±1.2	440.7±426.1	516.5±517.0	[27,3810]	[30,5333]
Cactus-fuzzy	12000	4	4.4±1.3	100.0±0.0	104.7±0.9	[100,100]	[101,108]
Cliques-fuzzy	12000	1	5.0±1.6	169.5±0.7	470.3±16.2	[165,170]	[404,534]

Table 10: **Synthetic dataset query patterns.** *Match*: whether instances are identified by exact subgraph isomorphism (exact) or approximate structural similarity (fuzzy). *Node attr.* / *Edge attr.*: feature type present on nodes / edges; ✗ = unlabeled (constant $\mathbf{1}$ feature).

Dataset	**#Classes**	**Node attr.**	**Edge attr.**	**Match**	**Pattern description**
Cactus40	6	✗	✗	exact	Cycles of lengths 3–8 (one class per length), each embedded in a cactus host graph (n=40).
Cactus200	6	✗	✗	exact	Same as Cactus40 with cycle lengths 11–16 and larger cactus hosts (n=200).
Cactus1000	6	✗	✗	exact	Same setup with cycle lengths 27–32 within larger hosts (n=1000).
Cliques100	1	✗	✗	exact	Complete graph K_5 (5 nodes, 10 edges). Injected into random bipartite hosts (n=50 per part) by replacing a randomly chosen node with a 5-clique and rewiring its incident edges to uniformly random clique members.
Cliques200	1	✗	✗	exact	Complete graph K_{10}; same injection procedure, larger bipartite host (n=100 per part).
Cliques400	1	✗	✗	exact	Complete graph K_{20} (20 nodes, 190 edges); same injection procedure, much larger bipartite host (n=200 per part).
ZINC12k	9	atom type	✗	exact	Nine frequent molecular subgraphs (3–7 nodes each) mined from ZINC12k by the GASTON algorithm (Nijssen and Kok, 2004) at minimum support 25% and max occurrences of 5 per graph. Mined patterns are illustrated in Figure 5. Ground-truth instance masks are the GASTON-reported node sets for each occurrence within each molecule.
Mol-Reddit	10	✗	✗	exact	Ten molecular structures (19–62 nodes) with all atom and bond features discarded (pure topology) are randomly sampled from the Molhiv dataset of OGB (Hu et al., 2020). Each structure is injected as a node-disjoint subgraph into a host graph drawn from REDDIT-MULTI-12K (Yanardag and Vishwanathan, 2015), with the number of attachment edges sampled from a Poisson distribution of mean 2. The host may contain unlabeled background motifs isomorphic to the injected patterns, requiring the model to distinguish true instances from background. Patterns are shown in Figure 6.
Cactus-fuzzy	4	binary	✗	fuzzy	Target graphs are cactus graphs containing cycles of lengths $k \in \{7, \ldots, 10\}$ as patterns. Each node receives a uniformly random label from $\{1, 2\}$. The canonical pattern for length k is a k-cycle with alternating labels 1-2-1-2-$\cdots$. A cycle is annotated as an instance if its graph edit distance to the canonical pattern is at most $\tau_{\text{GED}} = 3$; otherwise it is treated as background noise.
Cliques-fuzzy	1	✗	✗	fuzzy	The injection procedure follows the exact-match Cliques setting, but each injected k-clique is subsequently perturbed by randomly removing internal edges. Each injection is independently classified as an instance (with probability 50%) or as noise. Instances have 4 edges removed, while noise cliques have 8 edges removed, yielding a more aggressive perturbation. Only instances are annotated with ground-truth masks; noise cliques are present in the target graphs but not annotated, requiring the model to distinguish between mild and heavy perturbations.

Cactus graphs. A cactus graph is generated by sampling a random labeled tree on n nodes and injecting c edge-disjoint cycles of lengths drawn uniformly from a specified range, by identifying tree paths of the required length and closing them into cycles. Each injected cycle forms a chordless induced subgraph by construction. The pattern vocabulary contains one class per distinct cycle length. Node and edge features are omitted (constant feature $\mathbf{1}$).

Clique graphs. A host graph is a random bipartite graph $G(n_{\text{side}}, n_{\text{side}}, p)$ with n_{side} nodes on each side. Each clique injection replaces a randomly chosen, previously unexpanded node v with a k-clique, where v's neighbours are rewired to uniformly random nodes in the new clique, and v is deleted. Disconnected graphs are resampled. The single pattern class is the k-clique K_k.

ZINC12k. Frequent subgraph patterns are mined from the ZINC12k molecular benchmark using the GASTON algorithm (Nijssen and Kok, 2004) at minimum support 25% and a maximum of 5 occurrences per graph. Nine high-frequency subgraphs with 3–7 atoms are retained as query patterns; only atom type is preserved as a node attribute. Ground-truth instance masks are the GASTON-reported node sets for each occurrence within each molecule. Mined patterns are visualized in Figure 5.

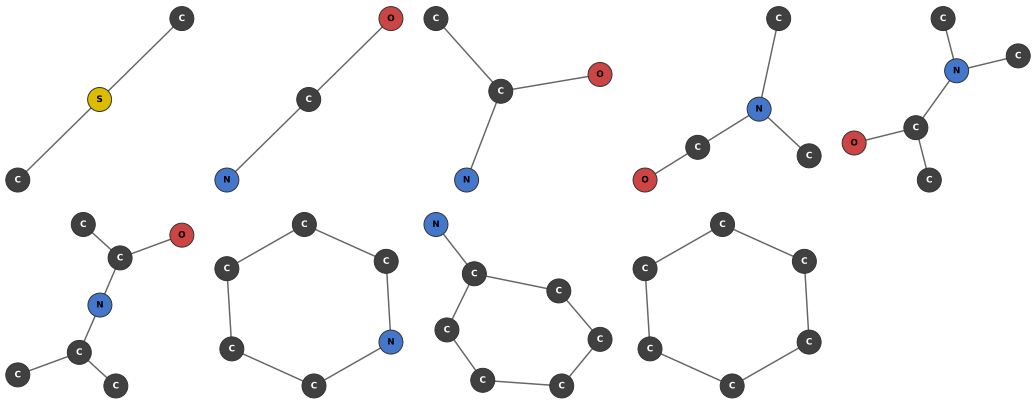


Figure 5: Query patterns for ZINC12k

Mol-Reddit. Molecular structures (query patterns) are injected into social network host graphs drawn from REDDIT-MULTI-12K (Yanardag and Vishwanathan, 2015). Ten molecular structures are randomly sampled from the Molhiv dataset of OGB (Hu et al., 2020) as pattern classes; their node features are discarded (constant $\mathbf{1}$), leaving only the topology for detection. Each injection adds a molecular subgraph as a node-disjoint component of the host graph, with $\sim$2 (following a Poisson distribution) random attachment edges connecting the molecule to the host. This dataset emulates the scenario of incomplete annotations: the original social network may contain graph motifs that are isomorphic to injected molecules, which are not labeled, so the model must distinguish true injected instances from background structure. The patterns are visualized in Figure 6.

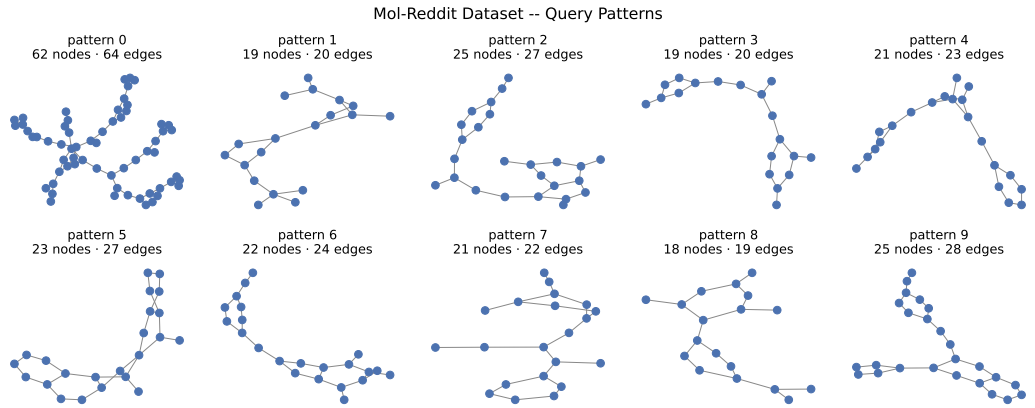


Figure 6: Query patterns for Mol-Reddit

Fuzzy Cactus. Each node is assigned a binary label drawn uniformly from $\{1, 2\}$. The pattern class for a cycle of length k is a *canonical* k-cycle with alternating labels 1-2-1-2-$\cdots$. An injected cycle is considered a *proper* instance if its node-label sequence differs from the canonical pattern (over all rotations and reflections) by at most τ_{GED} substitutions; otherwise it is treated as background noise. Four cycle-length classes (7–10) are used, with the graph size set to $n \approx 100$ nodes.

Fuzzy Cliques. The injection procedure follows the exact-match Cliques setting, but each clique is subsequently perturbed by removing a random subset of its internal edges, yielding an *approximate* clique pattern. A fraction of injections are designated as *noise* (with probability $p_{\text{noise}} = 0.5$) and receive more aggressive edge removal; only proper injections are annotated in the pattern file. The single pattern class is a fuzzy 8-clique (k=8) injected into bipartite host graphs.

F Implementation Details

F.1 Training setup

All models are trained with the AdamW optimiser (Loshchilov and Hutter, 2019) with weight decay 10^{-4}. The learning rate follows a linear warm-up over 100 steps to the peak learning rate, followed by cosine decay. Gradients are clipped to unit norm. All models are trained with PyTorch Lightning on a single GPU per run.

For the molecular task (ChEMBL12k), models are trained for up to 200 epochs with batch size 128. For synthetic tasks, epochs and batch sizes are adjusted per dataset owing to the wide range of graph sizes. All results are reported as mean $\pm$ std over 5 random seeds.

F.2 Hyperparameters

Table 11 lists the hyperparameters used for the main ChEMBL12k and ChEMBL experiments. Table 12 lists the search grid for the tuned hyperparameters; the other hyperparameters remain the same as for ChEMBL.

F.3 Computing resources

All experiments are run on a GPU cluster with NVIDIA A100 or H100 GPUs. A single ChEMBL12k run (200 epochs) completes in under 8 hours on one GPU. A full ChEMBL run (5 encoder layers, batch size 128) takes approximately one day on a single GPU. Synthetic dataset runs vary from a few hours (Cactus40, Cliques100) to over one day (Cactus1000, Mol-Reddit) depending on graph size.

G Additional Results

G.1 Full synthetic results

Table 13 reports all five evaluation metrics for every encoder–dataset combination.

G.2 Qualitative results on ChEMBL

We provide in Figure 7 a few examples from the ChEMBL dataset whose functional groups are successfully predicted by GraphDETR.

Table 11: Hyperparameters for the main GraphDETR configuration on ChEMBL12k and ChEMBL.

Hyperparameter	Value	Description
Model		
D (hidden dim)	256	Shared embedding dimension
Q (# queries)	40	Number of learnable query slots
encoder blocks	5	Number of graph encoder blocks for GCN/GIN/GraphGPS/NeuralWalker
RWSE dim K	20	Random-walk structural encoding steps in the graph encoder
Two-way transformer decoder		
L_{dec} (blocks)	3	Number of two-way transformer decoder blocks
h (attention heads)	8	Multi-head attention heads
d_{ff} (feedforward)	512	MLP hidden dimension
r (downsample rate)	2	Attention key/value dimension $= D/r$
NeuralWalker encoder		
walk length ℓ	20	Walk sequence length
walks per node (train)	1	Random walks sampled per node
walks per node (test)	5	Walks at inference time
sequence backend	Mamba	SSM sequence model
d_{conv}	9	Conv1d kernel width
d_{state}	16	Mamba SSM state size
window size W	8	Structural encoding context
bidirectional	✗	Forward-only walk processing
Loss		
λ_{mask}	4.0	Mask loss total weight
λ_{cut}	0.1	Graph cut penalty relative weight
$w_{\varnothing}$	0.1	Background class CE weight
Optimisation		
optimiser	AdamW	
learning rate	10^{-3}	Peak learning rate
weight decay	10^{-4}	AdamW weight decay
warmup steps	100	Linear warm-up steps
max epochs	200	ChEMBL12k training budget
gradient clip norm	1.0	
batch size	128	Graphs per batch

Table 12: Hyperparameter search grids for GraphDETR on synthetic datasets.

Hyperparameter	Search grid or values
Q (# queries)	$\{20, 40, 60\}$
λ_{cut}	$\{0.0, 0.001, 0.01, 0.1\}$
batch size	we adjust it based on the graph sizes in the dataset
NeuralWalker walk length	20 for Cactus40, ZINC12k, Cactus fuzzy, 5 for Cliques100, Cliques200, Cliques400, Cliques fuzzy, 50 for Mol-Reddit, 200 for Cactus200, 1000 for Cactus1000.

Table 13: **Full synthetic results.** All five metrics for every encoder–dataset combination (mean $\pm$ std over 5 seeds).

Dataset	Encoder	$\mathbf{AP}_{100}$	mAP	$\mathbf{Rec}_{100}$	mIoU	ExactMatch
Cactus40	GCN	$92.33_{\pm 0.56}$	$95.55_{\pm 0.19}$	$95.37_{\pm 0.26}$	$99.14_{\pm 0.05}$	$81.83_{\pm 2.30}$
	GIN	$92.98_{\pm 0.90}$	$95.98_{\pm 0.42}$	$95.76_{\pm 0.42}$	$99.21_{\pm 0.11}$	$83.77_{\pm 1.32}$
	GraphGPS	$96.51_{\pm 0.43}$	$97.84_{\pm 0.21}$	$97.82_{\pm 0.23}$	$99.57_{\pm 0.04}$	$89.63_{\pm 0.83}$
	NeuralWalker	$97.44_{\pm 0.44}$	$98.27_{\pm 0.24}$	$98.66_{\pm 0.29}$	$99.77_{\pm 0.04}$	$84.02_{\pm 1.79}$
Cactus200	GCN	$42.80_{\pm 3.16}$	$57.86_{\pm 5.04}$	$55.65_{\pm 3.87}$	$93.70_{\pm 0.94}$	$18.12_{\pm 10.59}$
	GIN	$41.51_{\pm 4.45}$	$52.94_{\pm 3.82}$	$53.47_{\pm 3.04}$	$93.84_{\pm 2.06}$	$9.20_{\pm 4.25}$
	GraphGPS	$83.67_{\pm 0.78}$	$92.17_{\pm 0.64}$	$90.44_{\pm 0.55}$	$98.96_{\pm 0.06}$	$84.12_{\pm 1.16}$
	NeuralWalker	$91.90_{\pm 2.41}$	$95.78_{\pm 1.60}$	$94.08_{\pm 1.55}$	$99.34_{\pm 0.17}$	$71.98_{\pm 13.66}$
Cactus1000	GCN	$10.46_{\pm 3.04}$	$17.75_{\pm 4.38}$	$18.44_{\pm 3.01}$	$89.25_{\pm 1.46}$	$0.00_{\pm 0.00}$
	GIN	$4.82_{\pm 7.30}$	$7.83_{\pm 11.85}$	$7.97_{\pm 11.74}$	$35.37_{\pm 48.44}$	$0.07_{\pm 0.15}$
	GraphGPS	$40.22_{\pm 0.68}$	$47.00_{\pm 0.26}$	$45.69_{\pm 0.29}$	$94.86_{\pm 0.31}$	$3.70_{\pm 0.40}$
	NeuralWalker	$42.20_{\pm 2.06}$	$61.51_{\pm 2.58}$	$57.81_{\pm 2.08}$	$94.06_{\pm 0.44}$	$0.32_{\pm 0.37}$
Cliques100	GCN	$90.94_{\pm 2.48}$	$94.03_{\pm 1.56}$	$93.53_{\pm 1.75}$	$98.73_{\pm 0.34}$	$69.90_{\pm 7.65}$
	GIN	$93.31_{\pm 0.74}$	$96.03_{\pm 0.47}$	$95.52_{\pm 0.57}$	$99.04_{\pm 0.11}$	$88.23_{\pm 4.25}$
	GraphGPS	$97.05_{\pm 1.60}$	$98.16_{\pm 0.79}$	$98.02_{\pm 1.17}$	$99.58_{\pm 0.25}$	$96.00_{\pm 4.74}$
	NeuralWalker	$96.15_{\pm 0.57}$	$97.40_{\pm 0.41}$	$97.71_{\pm 0.39}$	$99.49_{\pm 0.09}$	$95.93_{\pm 1.06}$
Cliques200	GCN	$90.33_{\pm 1.46}$	$95.21_{\pm 0.90}$	$93.91_{\pm 0.59}$	$99.11_{\pm 0.12}$	$86.47_{\pm 10.14}$
	GIN	$92.84_{\pm 2.20}$	$96.69_{\pm 1.04}$	$95.42_{\pm 1.85}$	$99.31_{\pm 0.27}$	$95.63_{\pm 2.39}$
	GraphGPS	$96.45_{\pm 1.17}$	$98.23_{\pm 0.50}$	$97.94_{\pm 0.88}$	$99.72_{\pm 0.14}$	$97.92_{\pm 1.64}$
	NeuralWalker	$93.48_{\pm 2.35}$	$97.17_{\pm 0.77}$	$96.08_{\pm 1.41}$	$99.45_{\pm 0.17}$	$97.18_{\pm 1.92}$
Cliques400	GCN	$92.51_{\pm 2.85}$	$95.91_{\pm 2.54}$	$95.11_{\pm 2.07}$	$99.37_{\pm 0.33}$	$87.28_{\pm 15.53}$
	GIN	$94.12_{\pm 1.51}$	$97.49_{\pm 0.54}$	$96.31_{\pm 0.58}$	$99.63_{\pm 0.06}$	$95.42_{\pm 1.13}$
	GraphGPS	$95.05_{\pm 3.23}$	$98.14_{\pm 0.99}$	$97.06_{\pm 2.34}$	$99.72_{\pm 0.28}$	$98.35_{\pm 2.25}$
	NeuralWalker	$92.58_{\pm 3.48}$	$97.53_{\pm 0.84}$	$95.77_{\pm 1.66}$	$99.60_{\pm 0.16}$	$97.78_{\pm 0.66}$
ZINC12k	GCN	$89.68_{\pm 0.88}$	$93.78_{\pm 0.47}$	$93.47_{\pm 0.52}$	$98.71_{\pm 0.14}$	$84.70_{\pm 2.34}$
	GIN	$92.86_{\pm 0.25}$	$95.60_{\pm 0.10}$	$95.33_{\pm 0.15}$	$99.11_{\pm 0.04}$	$88.73_{\pm 1.33}$
	GraphGPS	$95.28_{\pm 0.13}$	$97.15_{\pm 0.07}$	$97.08_{\pm 0.24}$	$99.38_{\pm 0.08}$	$92.58_{\pm 2.64}$
	NeuralWalker	$97.66_{\pm 0.27}$	$98.60_{\pm 0.16}$	$98.79_{\pm 0.18}$	$99.75_{\pm 0.04}$	$94.73_{\pm 1.33}$
Mol-Reddit	GCN	$65.86_{\pm 1.11}$	$85.05_{\pm 0.67}$	$76.04_{\pm 0.59}$	$96.51_{\pm 0.20}$	$93.17_{\pm 0.83}$
	GIN	$62.12_{\pm 1.31}$	$82.67_{\pm 0.92}$	$73.45_{\pm 0.93}$	$96.15_{\pm 0.24}$	$91.89_{\pm 0.82}$
	GraphGPS	$80.78_{\pm 1.32}$	$90.67_{\pm 1.08}$	$84.59_{\pm 1.44}$	$97.51_{\pm 0.28}$	$95.26_{\pm 1.12}$
	NeuralWalker	$89.76_{\pm 0.97}$	$95.26_{\pm 0.42}$	$93.12_{\pm 0.60}$	$98.99_{\pm 0.13}$	$97.30_{\pm 0.16}$
Cactus-fuzzy	GCN	$67.78_{\pm 1.17}$	$78.07_{\pm 1.08}$	$85.55_{\pm 0.92}$	$98.15_{\pm 0.07}$	$35.83_{\pm 2.91}$
	GIN	$76.19_{\pm 2.03}$	$84.19_{\pm 2.42}$	$89.56_{\pm 0.78}$	$98.70_{\pm 0.10}$	$45.15_{\pm 5.97}$
	GraphGPS	$84.86_{\pm 0.51}$	$91.34_{\pm 0.29}$	$93.57_{\pm 0.39}$	$99.15_{\pm 0.16}$	$79.95_{\pm 1.18}$
	NeuralWalker	$90.91_{\pm 2.88}$	$93.19_{\pm 2.92}$	$97.78_{\pm 0.35}$	$99.77_{\pm 0.04}$	$43.38_{\pm 15.89}$
Cliques-fuzzy	GCN	$95.05_{\pm 1.02}$	$96.81_{\pm 0.57}$	$96.83_{\pm 0.45}$	$99.45_{\pm 0.06}$	$90.82_{\pm 1.37}$
	GIN	$95.26_{\pm 0.89}$	$97.22_{\pm 0.46}$	$96.48_{\pm 0.62}$	$99.38_{\pm 0.15}$	$97.37_{\pm 0.85}$
	GraphGPS	$96.66_{\pm 0.86}$	$98.05_{\pm 0.48}$	$98.13_{\pm 0.68}$	$99.65_{\pm 0.13}$	$99.15_{\pm 0.55}$
	NeuralWalker	$93.94_{\pm 2.10}$	$96.58_{\pm 0.94}$	$96.32_{\pm 1.28}$	$99.35_{\pm 0.20}$	$96.30_{\pm 1.91}$

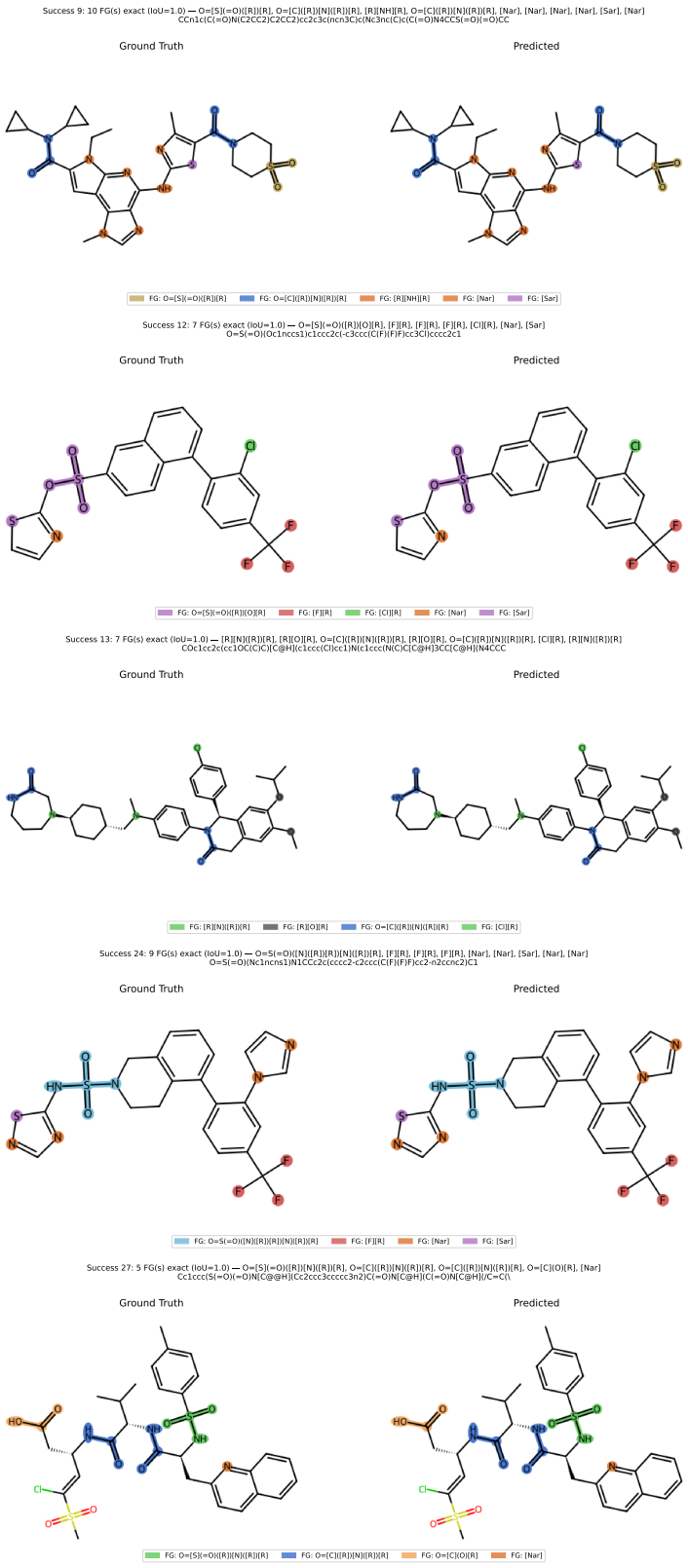


Figure 7: Visualization of ChEMBL examples whose functional groups are successfully predicted by GraphDETR.