Neural Graph Navigation for Intelligent Subgraph Matching: Technical Appendix

Anonymous submission

In this appendix, we provide a comprehensive elaboration of the methodologies, experimental details, and additional insights that support the findings presented in the main manuscript. The appendix is structured into the theoretical analysis of NeuGN (*Section A*), details of the proposed method (*Section B*), details of the experiments (*Section C*), and further discussions on limitation and future work (*Section D*). Furthermore, our source code is provided in the Supplementary Materials.

A Theoretical Analysis of NeuGN

This appendix provides the theoretical underpinning of NeuGN by formally proving its *completeness guarantee*, with a proof sketch presented below. Section A.1 demonstrates that the NeuGN-enhanced enumerator outputs exactly the same match set as any sound-and-complete baseline searcher (e.g., QSI, GQL, VF3, CECI, etc.) whose pruning routine obeys the standard safe–pruning requirement: no candidate that could still be extended to a full match is discarded before it is generated and tested, irrespective of the order in which its siblings are explored. The argument rests on two observations: (i) NeuGN is purely permutational, it reorders but never removes elements from the candidate list at each depth, so candidate coverage is preserved; (ii) classical order-robust pruning rules satisfy the safe–pruning property. Combining these facts, we prove that every root-to-leaf path yielding a valid match in the baseline search tree remains reachable when NeuGN dictates the visitation order, guaranteeing that no admissible match is lost even though the traversal order changes.

A.1 Completeness Guarantee

We prove that inserting NeuGN, which only reorders local candidate nodes, preserves the soundness and completeness of the underlying back-tracking enumerator.

Setup and Assumptions. Let $Q = (V_Q, E_Q)$ be the query graph, $G = (V_G, E_G)$ the data graph, and $\phi = (u_1, \ldots, u_{|V_Q|})$ a fixed visiting order of query vertices. At depth i the current partial injective mapping is $M_i = \{(u_j, v_j) \mid j \leq i\}$, and the next candidate multiset is $\mathcal{C}_{M_i}(u_{i+1})$. Depth-first search (DFS) expands candidates one by one; a full-depth leaf is a valid match.

Deterministic baseline order. The baseline enumerator explores every $\mathcal{C}_{M_i}$ in a fixed deterministic order π_0 (e.g. ascending vertex id).

Permutation policy. NeuGN replaces π_0 by a learned permutation $\pi_N(M_i)$ while keeping

(a) the same visiting order ϕ of query vertices and candidate generation,

(b) the same pruning predicate `prune`.

We adopt two explicit assumptions that hold for most classical algorithms (e.g. QSI, GQL, VF3, CECI).

(A1) Order-robust safe pruning. If a candidate (u_{i+1}, v) can still be extended to a full match, the pruning predicate `prune` never deletes the subtree rooted at that child *before* the child has been generated and tested, no matter in which order the other siblings of u_{i+1} are explored.

(A2) Admissible permutation. For every state M_i, $\pi(M_i)$ is a complete bijection on $\mathcal{C}_{M_i}(u_{i+1})$; i.e. all candidates appear exactly once. (Sampling top-k or dropping low-score elements would violate A2.)

Under assumptions (A1) and (A2), the search trees associated with both the baseline and NeuGN-enhanced enumeration procedures are finite and acyclic, thereby guaranteeing the termination of the depth-first search process.

Lemma 1 (Solution-Path Preservation) For any admissible permutation policy π, *every* the root-to-leaf path that produces a valid match under the baseline order π_0 is also generated under π.

Proof. Proceed by induction along a fixed baseline solution path $M_0 \to M_1 \to \ldots \to M_{|V_Q|}$. Assume the common prefix $M_0, \ldots, M_i$ has already been produced by the NeuGN order π. The next mapping on the path is $M_{i+1} = M_i \cup \{(u_{i+1}, v_{i+1})\}$. Because of (A2), v_{i+1} still appears in the permuted candidate list, and by (A1) the subtree below that child is not pruned before the candidate is tested. Hence DFS will visit M_{i+1} and the induction continues. □

Theorem 1 (Soundness and Completeness) If the baseline enumerator (with π_0) is sound and complete, then, under assumptions (A1) and (A2), the NeuGN-enhanced enumerator (with π_N) is also sound and complete.

Algorithm 1: NeuGN Framework for Subgraph Matching.

Input: A query graph Q and a data graph G.
Output: All the matches of Q into G.

1 $\boldsymbol{h_Q} \leftarrow$ **QSExtractor**$(\boldsymbol{Q})$
2 $C \leftarrow$ FilterNodes(Q, G)
3 $\varphi \leftarrow$ GenerateOrder(Q, G, C)
4 Enumerate$(Q, G, C, \varphi, \{\}, \boldsymbol{h_Q}, 0)$
5 **Function** *Enumerate(Q, G, C, φ, M, $\boldsymbol{h_Q}$, i)*:
6 **if** *Termination condition met* **then**
7 output TerminateFunc(Q, G, C, φ, M)
8 $u \leftarrow$ SelectNextQueryNode(Q, G, C, φ, M)
9 $C_M(u) \leftarrow$ ComputeLocalCandidates(Q, G, C, φ, M, i)
10 $\boldsymbol{Conf} \leftarrow$ **GGNavigator**$(\boldsymbol{Q}, \boldsymbol{G}, \boldsymbol{C}, \boldsymbol{h_Q}, \boldsymbol{\varphi}, \boldsymbol{M})$
11 $\boldsymbol{C_M(u)} \leftarrow$ **Sort**$(\boldsymbol{C_M(u)}, \boldsymbol{Conf})$
12 **foreach** $v \in C_M(u)$ **do**
13 PruningFunc(Q, G, C, φ, M)
14 Extend M by (u, v)
15 Enumerate$(Q, G, C, \varphi, M, i+1)$
16 Delete (u, v) from M

Proof. Soundness. Soundness is unchanged because all feasibility checks and the pruning predicate themselves are left intact. *Completeness.* Let $M^\star$ be any match output by the baseline, i.e. a leaf of its search tree. By Lemma A.1, the same solution path exists in the NeuGN search tree. Because PRUNE is safe according to (A1), no ancestor of $M^\star$ is removed before it is reached. Finite DFS therefore eventually visits $M^\star$, so the match is output as well.

□

Corollary 1. Any exact subgraph matcher whose pruning routine satisfies the order-robust safe pruning property (A1) can be combined with NeuGN without loss of completeness, irrespective of auxiliary data structures (e.g. failing sets, hash tables) that may depend on enumeration order.

This completes the proof.

B Details of the NeuGN Method

B.1 Overall Algorithm Description

The general framework of filtering-ordering-enumeration enhanced by NeuGN is outlined in Algorithm 1 in the main text, with a detailed description provided in this section.

Extracting Query Structure. The algorithm first extracts a structural signal h_Q of the query graph Q using the proposed **QSExtractor** (line 1). This signal serves as a compact representation of Q's structure patterns, enabling the GGNavigator to effectively align the evolving partial matches in the data graph G with the target substructure.

Filtering nodes. The algorithm then generates candidate sets by filtering nodes (line 2). Compared to traversing the entire graph, the filtering phase identifies potential matches for each query node and eliminates data graph nodes that cannot satisfy the matching constraints. The overall candidate set is denoted by C, and the candidates for a specific query node u are denoted by $C(u)$.

Matching Order. The algorithm then generates a matching order (line 3). A matching order φ of Q refers to a permutation of V_Q , determining the order in which nodes are explored during the search process. The index of vertex u in φ is denoted by $\varphi(u)$.

Enumeration Process. The algorithm finally conducts enumeration (line 5). The core enumeration process first computes the local candidate set $C_M(u)$ for the next query node (line 9). After obtaining $C_M(u)$, NeuGN employs the **GGNavigator** to score and rerank the candidates (lines 10–11). followed by extending partial matches (line 14), and recursively carries out the computations (line 15). The basic termination condition (line 6) is met when $i = |V_Q|$, indicating a complete match has been found. Pruning operations, which can eliminate invalid partial matches early, are applied during the enumeration (lines 13, 16). Some methods further optimize the process by using additional indices to avoid unnecessary backtracking.

B.2 (Semi-)Eulerian Node Path

To enable lossless and reversible graph serialization via (semi-)Eulerian paths, we employ the *eulerize* algorithm from NetworkX (Hagberg, Swart, and Schult 2008), which transforms an arbitrary connected undirected graph into an Eulerian graph by duplicating edges. The algorithm first identifies all vertices with odd degree. If none exist, the graph is already Eulerian. Otherwise, it computes the shortest paths between all pairs of odd-degree vertices and constructs an auxiliary complete graph G_p. A minimum-weight perfect matching in G_p identifies the optimal set of paths whose edges are duplicated, ensuring all vertices attain even degree with minimal edge addition. The Eulerization process has a polynomial time complexity of $O(|V|^3)$, which is dominated by the minimum-weight perfect matching computation on the set of odd-degree vertices.

Crucially, this Eulerization step is performed only on the query graph Q, which is typically small and fixed in size (e.g., $|V_Q| \ll |V_G|$). In contrast, the dominant cost of subgraph matching arises from the search over the data graph G, which has worst-case time complexity $O(|V_G|^{|V_Q|})$. Given that the Eulerization cost is polynomial in $|V_Q|$ and independent of $|V_G|$, it constitutes a negligible preprocessing overhead and is therefore fully acceptable within the overall computational framework.

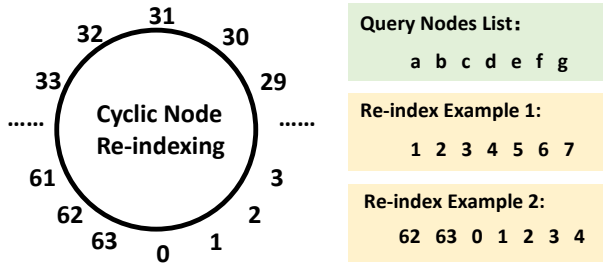

Figure S1: Illustration of cyclical re-indexing. Node indices are assigned in a circular manner over a fixed range $[0, N)$. A random offset r determines the starting position, and indices wrap around upon reaching the boundary.

B.3 Node Cyclic Re-indexing

In the main text, we describe a cyclic re-indexing scheme to mitigate biases arising from arbitrary node ordering in the Eulerian path serialization. A naive indexing strategy, assigning indices incrementally from 0 to $n-1$ based on first occurrence, can lead to imbalanced token training, where small-index tokens are over-represented while large-index tokens are under-sampled.

To address this, we adopt the *cyclic re-indexing* scheme introduced in (Zhao et al. 2025), which shifts the starting index by a random offset r sampled uniformly from $[0, N)$, where N is a predefined upper bound on node indices (e.g., $N = 64$). Specifically, an original index i is mapped to $i' = (i + r) \mod N$. This ensures that all index positions within the range $[0, N)$ are uniformly exposed during training, promoting balanced learning across the token embedding space.

As illustrated in Figure S1, the re-indexing process can be visualized as a circular shift on a ring of N positions. When the index sequence reaches the upper bound $N-1$, it wraps around to 0, forming a cycle. This circular structure guarantees that no index range is systematically favored, making the model robust to the arbitrary starting point of the Eulerian path.

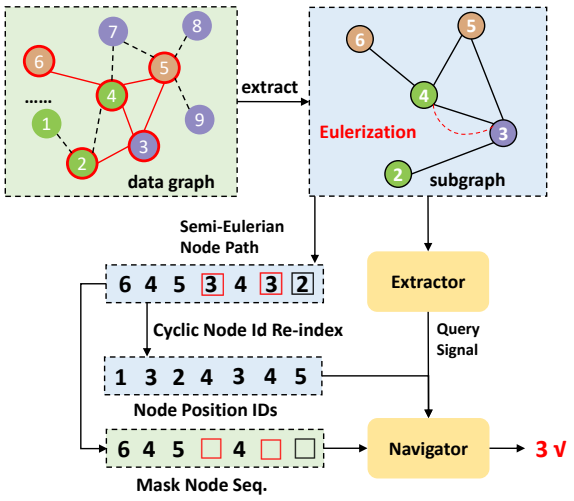

Figure S2: illustration of Training Data Preprocessing.

B.4 Data Preprocessing of Training

This section details the acquisition and processing of training data. As referenced in the main text, our self-supervised training directly utilizes subgraphs sampled from the data graph for masked node prediction, rather than relying on matches obtained by traditional subgraph matching algorithms. This design is motivated by two critical observations:

- **Computational Intractability**: Enumerating all isomorphic subgraphs in large data graphs is prohibitively expensive. For query graphs with $|V_Q| > 20$, single-match discovery may require seconds-level latency, creating a fundamental barrier for data-driven learning.
- **Training Scalability**: As reported in (Li et al. 2025), training neural order optimizers with exhaustive matches suffers from severe scalability issues: months-long training for larger queries with limited parallelization gains.

To overcome these constraints, we reformulate subgraph matching as intermediate state optimization: (1) Each sampled subgraph serves as a simulated query graph; (2) Masked node prediction approximates the next partial matching states; (3) This abstraction enables full batch parallelism while preserving structural learning objectives.

Figure S3 illustrates the complete workflow for a sampled subgraph (nodes with a red border in the data graph):

1. Eulerization: Convert subgraph to (semi-)Eulerian path (e.g., $6 \to 4 \to 5 \to 3 \to 4 \to 3 \to 2$).
2. Cyclical Re-indexing: Assign Node Position IDs via $i' = (i + r) \mod N$ with random offset r.
3. Masking Strategy:
   - Randomly mask k nodes (e.g., $\{2, 3\}$).
   - Select target node v_t (e.g., 3).
   - Replace v_t occurrences with `[Cls]` tokens.
   - Replace other masked nodes with `[Pad]` tokens.
4. Signal Extraction: Feed subgraph to QSExtractor to obtain h_Q.

Each training instance is formalized as a tuple: (Mask Node Sequence, Node Position IDs, h_Q, v_t) where:

- h_Q: Structural signal from QSExtractor.
- v_t: Ground-truth node ID for masked position.

The GGNavigator learns to predict v_t at `[Cls]` positions conditioned on h_Q and partial matches. This approach achieves substantially accelerated data generation compared to match-based methods while enabling GPU-parallelized batch processing.

B.5 Illustration of Mask Node Generation

This section provides a detailed explanation of the generative component in our graph generative navigation framework, specifically, how the prediction from the Mask Node Decoder is integrated with the subgraph matching completion process. As described in the *Euler-guided Mask Node Sequence Generation* section, the (semi-)Eulerian node path guides the generative completion of the Graph Template.

Suppose the matching order of the query graph, determined during the classical ordering phase, is $a \to c \to b \to d \to e$, and no matching failures occur during the enumeration phase. The step-by-step completion of the Graph Template proceeds as follows (see also Figure S3 for an illustrative example):

1. **Initialization**: Through Eulerization, we derive the (semi-)Eulerian node path and construct the initial Graph Template. At this stage, the position corresponding to the first unmatched query node a is marked with a class ID (denoted by a red ☐ in the Figure S3), while all other positions are filled with padding IDs (denoted by ☐).

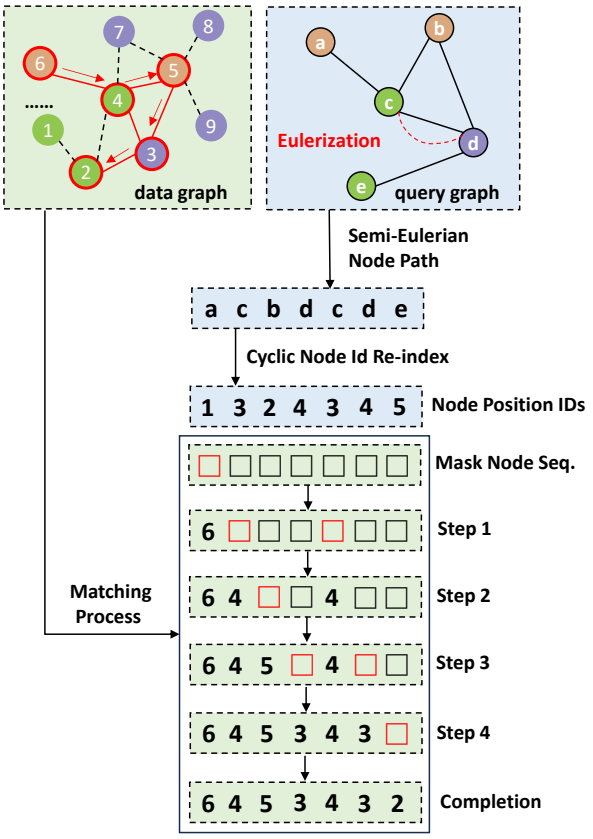

Figure S3: Step-by-step illustration of mask node generation in NeuGN. Red □: class ID (next to match); Black □: padding ID.

2. **First Matching Step**: The enumerator retrieves the set of local candidate nodes in the data graph for the query node a. The Neural Generative Navigator evaluates these candidates and selects the one with the highest confidence (e.g., node 6) and replaces the class token with its node ID. Subsequently, the class ID is moved to the positions corresponding to the next unmatched query node, c. Since c appears twice in the (semi-)Eulerian path due to edge duplication during Eulerization, both of its positions are assigned the class IDs.
3. **Second Matching Step**: The enumerator now fetches the local candidates for query node c. The navigator selects the most promising candidate (e.g., node 4) and replaces **all** class tokens with the node 4's ID. The class ID is then advanced to the positions of the next unmatched node, b.
4. **Iterative Completion**: This process continues iteratively: at each step, the navigator uses the current partial matching state and structural context to predict the most likely candidate for the node marked by the class token. The selected node ID replaces the class token, and the class token advances to the next relevant positions in the (semi-)Eulerian sequence.

This generation process continues until all padding and class tokens are replaced by actual node IDs from the data graph, resulting in a complete and valid subgraph matching. The integration of neural guidance within this template-filling process enables NeuGN to prioritize high-probability enumeration paths intelligently, significantly reducing unnecessary search.

C Details of the Experiments

This section provides a comprehensive overview of the experimental setup and results. We first describe the six graph datasets used in our evaluation, followed by implementation details for both baseline methods and NeuGN. We then present additional experimental results to support our approach. Furthermore, we report the inference time of our proposed plugin to assess the computational efficiency of the neuro-heuristic framework.

C.1 Datasets

We evaluate our method on six real-world graph datasets spanning diverse domains: social networks (Hamster, LastFM, YouTube), knowledge graphs (NELL), academic collaboration networks (DBLP), and web-based information systems (WikiCS). These datasets exhibit varied topological structures and semantic characteristics, enabling comprehensive evaluation of subgraph matching performance across different graph regimes.

The details of each dataset are as follows:

- *Hamster* (Kunegis 2013) is a social network from the website hamsterster.com, where nodes represent users and edges denote friendship or family relationships. Node classes indicate the type of hamster associated with each user.
- *LastFM* (Rozemberczki and Sarkar 2020) captures a social network of LastFM users from Asian countries, collected via the public API. Nodes are users, edges represent mutual follower relationships, and node classes correspond to users' home countries.
- *WikiCS* (Mernyei and Cangea 2020) is a citation-based graph of Computer Science articles, where nodes represent papers and edges are hyperlinks between them. Node classes denote different subfields within computer science.
- *NELL* (Carlson et al. 2010; Yang, Cohen, and Salakhudinov 2016) is derived from the Never-Ending Language Learner system, which continuously extracts facts from web text. The graph consists of entities as nodes and relational facts as edges, with node classes indicating semantic categories.
- *DBLP* (Yang and Leskovec 2012a) is a co-authorship network where nodes represent authors and edges indicate joint publications. Node classes correspond to four major research areas in computer science, inferred from publication venues.
- *YouTube* (Yang and Leskovec 2012b) is a social network of YouTube users, with edges representing friendship relations. Node classes are defined by user-created group memberships, reflecting community interests such as music, gaming, and education.

C.2 Implementation Details

Implementation of Baselines. To evaluate the proposed NeuGN framework, we adopt eight advanced subgraph matching methods for integration, encompassing both traditional heuristic-based and neural-enhanced approaches: VF3 (Carletti et al. 2017), QSI (Shang et al. 2008), GQL (Bi et al. 2016), CECI (Bhattarai, Liu, and Huang 2019), CFL (Bi et al. 2016), CaLiG (Yang et al. 2023), RLQVO (Wang et al. 2022), and RSM (Li et al. 2025). Among them, VF3, QSI, GQL, CECI, CFL, and CaLiG are traditional heuristic-based methods, while RLQVO and RSM are neural-enhanced approaches.

To ensure a fair comparison, for traditional rule-based methods, we use the implementations integrated in (Zhang et al. 2024) and enforce single-threaded execution across all methods. For learning-enhanced approaches, we use their publicly available source codes and follow the official hyperparameter settings provided by the original authors.

Implementation Settings. We summarize the key hyperparameters of the proposed NeuGN framework architecture and training process as follows. The model is trained for 1000 epochs with a batch size of 128. In each epoch, the query subgraphs are sampled by performing random walks that visit 5 to 19 distinct nodes starting from each node in the data graph, and the induced subgraph over the visited nodes is extracted as the query. This sampling strategy ensures that larger queries (e.g., 20, 24, and 32 nodes) present in the test set do not appear during training. The token embedding dimension is set to 256. The Transformer decoder consists of 4 layers, each with 8 attention heads, and the hidden dimension of the feed-forward network (FFN) layers is 1024. For the Hamster and LastFM datasets, the decoder is reduced to 2 layers to accommodate their smaller graph scales and lower structural complexity. We optimize the model using the Adam optimizer with a learning rate of 5×10^{-4} and employ an exponential learning rate scheduler with a decay factor of $\gamma = 0.999$.

Moreover, our experiments were conducted on a high-performance server equipped with Intel(R) Xeon(R) Gold 6342 CPUs@2.80 GHz, along with NVIDIA A40 GPUs (48GB VRAM). Specifically, the offline training phase was performed using a 4×A40 GPU configuration in parallel, while the online subgraph matching inference was carried out on a single A40 GPU.

C.3 Additional Experimental Results

Efficiency Analysis of Batched Neural Inference. Table S1 presents batched inference latency measurements (in milliseconds) on a single NVIDIA A40 GPU with fixed sequence length 64, selected for its representativeness across typical subgraph matching scenarios. All latency values are averaged over 10,000 independent measurement runs to ensure statistical reliability. All reported results incorporate a inference-time optimization: the removal of the softmax layer from the output module. This modification exploits the order, preserving property of linear scoring, specifically, $\text{argsort}(\mathbf{W}h_{[\text{Cls}]}) \equiv \text{argsort}(\text{softmax}(\mathbf{W}h_{[\text{Cls}]}))$, which ensures identical candidate rankings while eliminating the computational overhead of exponentiation and normalization. The results demonstrate practical operational efficiency: after amortization of latency through batch processing (BS = 16), the inference of the per query is completed in a submillisecond time for most datasets (0.06 to 0.75 ms). Even for the YouTube graph (1.13M nodes), latency remains well below 3ms (2.43ms), comfortably satisfying real-time constraints for applications. This efficiency stems from NeuGN's GPU-accelerated design, where batching distributes fixed computational costs across multiple queries. The observed latency progression, increasing predictably with graph node vocabulary size, aligns with architectural expectations, confirming measurement validity.

Dataset	Hamster	LastFM	WikiCS	NELL	DBLP	YouTube
BS=1	0.96	0.96	1.52	1.55	2.03	3.71
BS=4	1.01	1.04	1.64	2.04	4.33	11.12
BS=16	1.03	1.24	1.99	3.84	12.05	38.91
PQ(B=4)	0.25	0.26	0.41	0.51	1.08	2.78
PQ(B=16)	0.06	0.08	0.12	0.24	0.75	2.43

Table S1: Batched inference latency (ms) of GGNavigator with sequence length=64. Total latency for batch sizes and per-query amortized latency.

Training Efficiency Analysis. Table S2 presents the per-epoch training time across datasets using our full-node sampling strategy, executed on a parallelized 4×NVIDIA A40 GPU configuration. This design choice ensures comprehensive coverage of local structural patterns by treating each node as an anchor point for query subgraph generation, thereby preventing any structural motifs from being overlooked during training. The training time exhibits a near-linear relationship with graph size, increasing from 0.02 minutes on Hamster (2.4K nodes) to 19.30 minutes on YouTube (1.13M nodes). The observed time complexity aligns with the theoretical $O(|V|)$ expectation of our sampling methodology, where $|V|$ represents the number of vertices in the data graph.

Dataset	Hamster	LastFM	WikiCS	NELL	DBLP	YouTube
Time (min)	0.02	0.05	0.11	0.38	3.09	19.30

Table S2: Per-epoch training time (minutes) across datasets

Details and Additional Results of Early Convergence Acceleration (RQ5). This section provides supplementary details to the early convergence experiment in the main text. All NeuGN inference operations were executed on a single NVIDIA A40 GPU with the following constrained configuration: batch size fixed at 4, navigation depth set to 16, and inference iterations limited to 24 to avoid computational overhead. Extended results demonstrating acceleration performance are presented in Table S3.

	Hamster	LastFM	WikiCS	NELL	DBLP	YouTube
GQL	4.29E+05	8.22E+05	3.06E+05	4.58E+05	8.63E+05	1.83E+05
+NeuGN	5.02E+05	1.14E+06	4.67E+05	5.98E+05	9.02E+05	2.21E+05
Improv.	**17.0%**	**39.0%**	**52.4%**	**30.5%**	**4.5%**	**20.7%**
CFL	5.42E+06	6.50E+06	3.41E+06	6.02E+06	1.19E+07	4.82E+06
+NeuGN	5.70E+06	7.05E+06	5.30E+06	7.15E+06	1.22E+07	5.12E+06
Improv.	**5.2%**	**8.5%**	**55.6%**	**18.7%**	**2.1%**	**6.2%**
CECI	5.15E+06	5.04E+06	4.93E+06	4.79E+06	7.47E+06	5.07E+06
+NeuGN	5.62E+06	5.94E+06	6.79E+06	5.76E+06	7.74E+06	5.59E+06
Improv.	**9.2%**	**17.8%**	**37.7%**	**20.2%**	**3.6%**	**10.3%**
CaLiG	6.58E+06	5.54E+06	4.43E+06	4.46E+06	9.89E+06	2.22E+06
+NeuGN	6.78E+06	6.28E+06	8.50E+06	6.09E+06	1.01E+07	2.48E+06
Improv.	**3.0%**	**13.3%**	**91.6%**	**36.5%**	**2.0%**	**11.6%**

Table S3: Additional Results of RQ5.

D Discussion

D.1 Limitation

NeuGN improves subgraph matching only through permutational navigation: it reorders candidate vertices but never prunes them, thereby preserving completeness at the cost of leaving the search tree size essentially unchanged. Although a neural classifier that eliminates hopeless branches could in theory yield greater speed-ups, integrating such pruning remains impractical because any false-negative decision would break completeness and undermine the exact-matching guarantee.

D.2 Future Work

Graph Tokenization for Web-Scale Deployment. To effectively alleviate the vocabulary explosion problem encountered in web-scale graphs ($>$ 100M nodes), we plan to explore hierarchical tokenization strategies for NeuGN based on METIS partitioning (Karypis and Kumar 1998a,b), using dual-token identifiers $[\text{PartitionID}, \text{LocalNodeID}]$. This approach compresses the node vocabulary size from $O(|V|)$ to $O(\sqrt{|V|})$, significantly reducing model memory footprint, while aiming to preserve essential topological context within and across partitions.

Semantic-aware Matching for Complex Structures. Classical subgraph matching algorithms are limited in handling high-dimensional semantic graphs (e.g., text-attributed academic concept networks) due to their reliance on rigid, rule-based heuristics. In contrast, NeuGN's inherent semantic awareness opens a pathway toward future cross-modal subgraph matching, aligning topological patterns with unstructured text or visual attributes.

References

Bhattarai, B.; Liu, H.; and Huang, H. H. 2019. Ceci: Compact embedding cluster index for scalable subgraph matching. In *Proceedings of the 2019 International Conference on Management of Data*, 1447–1462.

Bi, F.; Chang, L.; Lin, X.; Qin, L.; and Zhang, W. 2016. Efficient subgraph matching by postponing cartesian products. In *Proceedings of the 2016 International Conference on Management of Data*, 1199–1214.

Carletti, V.; Foggia, P.; Saggese, A.; and Vento, M. 2017. Challenging the time complexity of exact subgraph isomorphism for huge and dense graphs with VF3. *IEEE transactions on pattern analysis and machine intelligence*, 40(4): 804–818.

Carlson, A.; Betteridge, J.; Kisiel, B.; Settles, B.; Hruschka, E.; and Mitchell, T. 2010. Toward an architecture for never-ending language learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 24, 1306–1313.

Hagberg, A.; Swart, P. J.; and Schult, D. A. 2008. Exploring network structure, dynamics, and function using NetworkX. Technical report, Los Alamos National Laboratory (LANL), Los Alamos, NM (United States).

Karypis, G.; and Kumar, V. 1998a. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing*, 20(1): 359–392.

Karypis, G.; and Kumar, V. 1998b. Multilevelk-way partitioning scheme for irregular graphs. *Journal of Parallel and Distributed computing*, 48(1): 96–129.

Kunegis, J. 2013. Konect: the koblenz network collection. In *Proceedings of the 22nd international conference on world wide web*, 1343–1350.

Li, Z.; Dou, Y.; Li, Y.; Chen, X.; and Zhang, C. 2025. RSM: Reinforced Subgraph Matching Framework with Fine-grained Operation based Search Plan. In *Proceedings of the Eighteenth ACM International Conference on Web Search and Data Mining*, 475–483.

Mernyei, P.; and Cangea, C. 2020. Wiki-cs: A wikipedia-based benchmark for graph neural networks. *arXiv preprint arXiv:2007.02901*.

Rozemberczki, B.; and Sarkar, R. 2020. Characteristic Functions on Graphs: Birds of a Feather, from Statistical Descriptors to Parametric Models. In *Proceedings of the 29th*

ACM International Conference on Information and Knowledge Management (CIKM '20), 1325–1334. ACM.

Shang, H.; Zhang, Y.; Lin, X.; and Yu, J. X. 2008. Taming verification hardness: an efficient algorithm for testing subgraph isomorphism. *Proceedings of the VLDB Endowment*, 1(1): 364–375.

Wang, H.; Zhang, Y.; Qin, L.; Wang, W.; Zhang, W.; and Lin, X. 2022. Reinforcement learning based query vertex ordering model for subgraph matching. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*, 245–258. IEEE.

Yang, J.; and Leskovec, J. 2012a. Defining and evaluating network communities based on ground-truth. In *Proceedings of the ACM SIGKDD workshop on mining data semantics*, 1–8.

Yang, J.; and Leskovec, J. 2012b. Defining and evaluating network communities based on ground-truth. In *Proceedings of the ACM SIGKDD workshop on mining data semantics*, 1–8.

Yang, R.; Zhang, Z.; Zheng, W.; and Yu, J. X. 2023. Fast continuous subgraph matching over streaming graphs via backtracking reduction. *Proceedings of the ACM on Management of Data*, 1(1): 1–26.

Yang, Z.; Cohen, W.; and Salakhudinov, R. 2016. Revisiting semi-supervised learning with graph embeddings. In *International conference on machine learning*, 40–48. PMLR.

Zhang, Z.; Lu, Y.; Zheng, W.; and Lin, X. 2024. A Comprehensive Survey and Experimental Study of Subgraph Matching: Trends, Unbiasedness, and Interaction. *Proceedings of the ACM on Management of Data*, 2(1): 1–29.

Zhao, Q.; Ren, W.; Li, T.; Liu, H.; He, X.; and Xu, X. 2025. GraphGPT: Generative Pre-trained Graph Eulerian Transformer. In *Forty-second International Conference on Machine Learning*.