Enumerating Graph Pattern Matches with ML Oracles

WENFEI FAN, Beihang University, China, Shenzhen Institute of Computing Sciences, China, and University of Edinburgh, United Kingdom
YIXUAN LUO, Beihang University, China
PING LU*, Beihang University, China
XIAOKE ZHU, Shenzhen Institute of Computing Sciences, China

This paper presents an ML-based method for SubIso, the graph pattern matching problem. Given a graph G and a pattern Q, SubIso aims to enumerate all subgraphs of G that are isomorphic to Q. We show that SubIso is EnumP-complete, where EnumP is the enumeration counterpart of NP. We then study revisions $\mathsf{VF3}_M$ of the classical VF3 algorithm by incorporating an ML oracle $\mathcal{M}$. The model $\mathcal{M}$ predicts candidate matches, while $\mathsf{VF3}_M$ verifies them, thereby reducing costly backtracking. We examine whether $\mathsf{VF3}_M$ is (a) output-polynomial, *i.e.,* its runtime is polynomial in the sizes of input and output; (b) consistent, *i.e.,* its accuracy reaches 100% with error-free predictions from $\mathcal{M}$; and (c) β-robust, *i.e.,* it guarantees accuracy of at least β even with arbitrarily bad predictions.

We establish several results, positive and negative. On the negative side, we show that it is impossible for $\mathsf{VF3}_M$ to be both output polynomial and β-robust with a positive constant β unless P = fewP, a problem as hard as P = NP. On the positive side, we develop three versions of $\mathsf{VF3}_M$ that are (a) consistent, and (b) either 1-robust or output polynomial with a high probability. We also train a model $\mathcal{M}$ for $\mathsf{VF3}_M$. Using real-life and synthetic data, we show that $\mathsf{VF3}_M$ is up to 15.75×–21× faster than VF3, with F1-score 0.98.

CCS Concepts: • **Information systems → Graph-based database models; Information retrieval query processing**.

Additional Key Words and Phrases: Graph pattern matching; Machine learning; Output polynomial; Competitive ratio; Consistency; Robustness

ACM Reference Format:
Wenfei Fan, Yixuan Luo, Ping Lu, and Xiaoke Zhu. 2026. Enumerating Graph Pattern Matches with ML Oracles. *Proc. ACM Manag. Data* 4, 1 (SIGMOD), Article 34 (February 2026), 28 pages. https://doi.org/10.1145/3786648

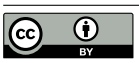
1 Introduction

Graph pattern matching takes as input a pattern Q and a graph G, and computes the set $Q(G)$ of all subgraphs of G isomorphic to Q. It is fundamental in graph databases, biochemistry, computer vision, social network analysis, and fraud detection, among others. We refer to this problem as SubIso. Its decision version asks whether Q has a match in G, which is NP-complete [45]. Thus, unless P = NP, no exact polynomial-time (PTIME) algorithm for SubIso exists.

For intractable problems, research on PTIME algorithms has largely focused on their combinatorial optimization versions, seeking optimal (maximum or minimum) solutions via four main

*Corresponding author.

Authors' Contact Information: Wenfei Fan, Beihang University, China and Shenzhen Institute of Computing Sciences, China and University of Edinburgh, United Kingdom, wenfei@inf.ed.ac.uk; Yixuan Luo, Beihang University, China, luoyx@act.buaa.edu.cn; Ping Lu, Beihang University, China, luping@buaa.edu.cn; Xiaoke Zhu, Shenzhen Institute of Computing Sciences, China, zhuxk@sics.ac.cn.



ACM 2836-6573/2026/2-ART34
https://doi.org/10.1145/3786648

approaches: (1) approximation, which guarantees solutions within a constant factor of the optimum; (2) heuristics, which often perform well in practice but lack provable guarantees; (3) randomization, which achieves fast expected runtimes at the cost of a small failure probability; and (4) parameterization, which yields PTIME algorithms when certain input parameters are fixed [44].

While the decision version of SubIso is hard, the enumeration problem is even more challenging. In practice, we often need all answers to a problem; for example, evaluating an SQL query requires computing every result in the dataset, not just checking for existence or optimality. Likewise, for SubIso, we must find all matches of a pattern Q in a graph G to, *e.g.,* answer a SPARQL query or compute rule confidence [43]. However, SubIso is inherently exponential, with up to $O(|G|^{|Q|})$ matches in the worst case. Moreover, algorithmic techniques developed for decision or optimization problems do not extend well to enumeration.

Does there exist an efficient and accurate algorithm for SubIso?

An ML-based approach. We explore SubIso by employing machine learning (ML) models as oracles. For proof of concept (PoC), we revise the classical VF3 algorithm [28, 29] by integrating an ML model $\mathcal{M}$ that predicts whether a partial match ρ of Q is valid, *i.e.,* extendable to a full match in G. Guided by $\mathcal{M}$, $\mathsf{VF3}_M$ enumerates matches recursively as in VF3, but can skip costly backtracking when $\mathcal{M}$ deems ρ invalid. We call this family of revisions $\mathsf{VF3}_M$.

Intuitively, we want to unify algorithmic methods and ML predictions. On the one hand, SubIso algorithms have been studied for decades. On the other hand, ML models have proven effective in a variety of applications. Hence, it is natural to combine the two and take advantage of both. Moreover, even when we cannot find a good model $\mathcal{M}$ now, when ML techniques evolve, the quality of $\mathcal{M}$ predictions will improve and $\mathsf{VF3}_M$ will get more accurate.

The idea is not entirely new. Algorithms with ML predictions have been studied for decision problems [24, 25, 33, 35, 66, 73, 80, 81, 86, 89, 108]. There has also been work on learning meta-algorithms for optimization problems [63]. However, the prior methods (a) do not carry over to SubIso, an enumeration problem, (b) rely on ML alone, instead of unifying algorithmic methods and ML predictions, and (c) guarantee neither the accuracy nor the efficiency (see below).

Challenges. It is nontrivial to develop "good" $\mathsf{VF3}_M$ algorithms. We advocate metrics for measuring their accuracy and efficiency.

(1) Output polynomial. Ideally, we want $\mathsf{VF3}_M$ to *enumerate matches* in PTIME in the sizes of the input and output (assuming that applying $\mathcal{M}$ is in PTIME after $\mathcal{M}$ is trained, as commonly found in practice). An enumeration problem with such an exact algorithm is considered "tractable" and is in the complexity class OutputP [27]. However, we show that SubIso is EnumP-complete, where EnumP [105] is the class of enumeration problems for which the correctness of a candidate solution can be *checked* in PTIME, and is the enumeration equivalent of NP. Hence, SubIso is one of the hardest problems in EnumP. It is known that EnumP $\neq$ OutputP unless P = NP [27].

(2) Consistency and robustness. Obviously, the accuracy of $\mathsf{VF3}_M$ is impacted by the rate of false positives (FPs) and false negatives (FNs) of its embedded ML model $\mathcal{M}$. This said, we want $\mathsf{VF3}_M$ to be both (a) consistent, *i.e.,* it accuracy approaches 100% when the predictions of $\mathcal{M}$ are error-free, and (b) β-robust for a bound β, *i.e.,* its accuracy is at least β even when $\mathcal{M}$ gives arbitrarily bad predictions. That is, the more accurate $\mathcal{M}$ is, the more reliable $\mathsf{VF3}_M$ is. Moreover, $\mathsf{VF3}_M$ still performs well even when $\mathcal{M}$ is inaccurate.

Does there exist $\mathsf{VF3}_M$ such that it is consistent, β-robust (for a positive constant β) and output polynomial at the same time?

Contributions and organization. This paper aims to develop a better understanding of these issues. We establish several results about SubIso; some results are positive, while some are negative.

(1) EnumP-*hardness* (Section 2). We show that SubIso is EnumP-complete. Thus unless P = NP, one cannot expect to find an exact algorithm for SubIso that is *output polynomial*, *i.e.,* its cost can be expressed as a polynomial in the sizes of input and output. On the positive side, if we find an output-polynomial enumeration algorithm for SubIso [97], then all the enumeration problems in EnumP will have a "tractable" accurate algorithm, since all such problems can be reduced to SubIso with PTIME parsimonious reductions, *i.e.,* transformations that preserve the number of solutions [85].

(2) Algorithms $\mathsf{VF3}_M$ (Section 3). We then develop three $\mathsf{VF3}_M$ algorithms for SubIso by incorporating an ML model $\mathcal{M}$ into VF3, denoted by $\mathsf{VF3}_M^N$, $\mathsf{VF3}_M^O$ and $\mathsf{VF3}_M^D$, respectively, adopting different backup plans when the predictions of $\mathcal{M}$ may be bad. $\mathsf{VF3}_M^N$ opts to completely trust $\mathcal{M}$ and follows its predictions without backup. $\mathsf{VF3}_M^O$ trusts $\mathcal{M}$ only if its confidence is high, and resorts to VF3 as a backup otherwise. $\mathsf{VF3}_M^D$ conducts deep checking if $\mathcal{M}$ predicts false, to further reduce FNs and improve the robustness. The three strive for efficiency, accuracy and their balance, respectively.

(3) Performance guarantees (Section 4). We show a negative result: unless P = fewP, no algorithm for SubIso can be both output-polynomial and β-robust for any positive constant β; here fewP is the class of problems recognizable by a nondeterministic Turing machine in PTIME and having polynomially-bounded number of solutions [9]. We know that fewP = P is as hard as P = NP [37]. This result rules out attempts to develop algorithms with both properties simultaneously. It applies only to EnumP-hard problems, not to the enumeration of all NP-complete problems (see Section 4.1).

On the positive side, we show that: (a) all three $\mathsf{VF3}_M$ algorithms have precision 1, *i.e.,* every solution they output is a correct match for SubIso, with no false positives; (b) all are consistent, achieving 100% accuracy when $\mathcal{M}$ is error-free; (c) $\mathsf{VF3}_M^O$ and $\mathsf{VF3}_M^D$ are "almost" 1-robust with recall = 1 when $\mathcal{M}$ predictions have high confidence; and (d) with high probability, $\mathsf{VF3}_M^N$ is "almost" output-polynomial and β-robust, where β depends on output size. These results guide the development of practical SubIso algorithms, and as the ML model improves, their accuracy should approach perfection.

(4) ML model $\mathcal{M}$ (Section 5). We train an ML model $\mathcal{M}$ for $\mathsf{VF3}_M$ algorithms. Departing from previous models for subgraph matching or counting, it predicts whether a partial match is valid. We adopt IDGNN [121] for vertex embedding, develop a partition-based strategy to embed patterns and graphs and encode the given partial match, make a prediction using the order-embedding space, and return a confidence of the prediction via a multilayer perceptron. Moreover, we enrich training data to ensure the robustness of $\mathcal{M}$.

(5) Effectiveness (Section 6). Using real-life and synthetic graphs, we experimentally find the following. (a) By unifying algorithmic methods and ML predictions, $\mathsf{VF3}_M^N$, $\mathsf{VF3}_M^O$ and $\mathsf{VF3}_M^D$ speed up VF3 by 10.35×, 9.63× and 8.40× on average, respectively, up to 21×, 19.38× and 15.75×. (b) The $\mathsf{VF3}_M$ algorithms are accurate. They are consistent (*i.e.,* their precision is constantly 1), and their average F1 scores are 0.44, 0.95 and 0.98, respectively, up to 0.62, 0.99 and 0.99. (c) Algorithms $\mathsf{VF3}_M^O$ and $\mathsf{VF3}_M^D$ are robust: their F1 scores are above 0.9, even when the embedded ML model has error rate 0.5. (d) Our ML model $\mathcal{M}$ is 15% more accurate than subgraph matching and counting models on average, up to 29%.

We discuss related work in Section 7 and future work in Section 8.

Beyond SubIso, this work offers insights for developing practical algorithms for enumeration problems in EnumP (by Theorem 1), where effective methods remain largely undeveloped.

2 SubIso, EnumP, and OutputP

This section reviews the SubIso problem and complexity classes EnumP and OutputP. We show that SubIso is EnumP-complete.

Graphs. Assume a countably infinite set Γ of symbols for labels. We consider *labeled directed graphs* $G = (V, E, L)$, where (a) V is a finite set of vertices, (b) $E \subseteq V \times \Gamma \times V$ is a finite set of edges in which (v_1, l, v_2) is an edge labeled $l \in \Gamma$ from v_1 to v_2, and (c) L is a function such that for each vertex $v \in V$, $L(v) \in \Gamma$ is its label.

Graph pattern matching. We next present SubIso.

Patterns. A *graph pattern* is a connected graph $Q = (V_Q, E_Q, L_Q)$, where (a) V_Q (resp. E_Q) is a finite set of pattern vertices (resp. edges), and (b) L_Q assigns a label $L_Q(u)$ to each pattern vertex $u \in V_Q$.

Matching. A *full match* h of pattern Q in a graph G is a bijective mapping from V_Q to V' of a subgraph $G' = (V', E', L')$ of G such that (a) for each vertex u in V_Q, $L_Q(u) = L'(h(u))$; and (b) $e = (u, l, u')$ is an edge in E_Q iff $e' = (h(u), l, h(u'))$ is an edge in E' for label l in Γ.

SubIso. The numeration problem SubIso is stated as follows.

- *Input*: A graph pattern Q and a graph G.
- *Output*: The set $Q(G)$ of all full matches of Q in G.

The set $Q(G)$ may have $O(|G|^{|Q|})$ many matches of Q in G.

Partial match. A *partial match* of Q in G is a bijective mapping ρ from a *subgraph* of Q to a subgraph of G as above. Match ρ is called *valid* if it can be extended to be a full match h of Q in G such that for each $u \in V_Q$, $\rho(u) = h(u)$ if $\rho(u)$ is defined. Full matches are a special case of partial matches that cannot be further extended.

Example 1: For graph G and pattern Q in Figure 1, there exists a unique full match of Q in G, namely, the leftmost subgraph of G. Consider two partial matches: (1) ρ_1, defined by $h(u_i) = v_i$ for $i \in [1, 5]$ (colored in red), is valid, as it can be extended to the full match by adding $h(u_6) = v_6$ and $h(u_7) = v_7$; and (2) ρ_2, defined by $h(u_1) = v_1$ and $h(u_i) = v_{i+6}$ for $i \in [2, 5]$ (blue circles), is invalid: when mapping u_5 to v_{11}, vertex u_6 has no valid match in G, since all neighbors of v_{11} are already used in ρ_2. As full matches require bijective mappings, each pattern vertex in Q must map to a distinct vertex in G. □

Complexity classes. We consider two complexity classes EnumP and OutputP [27]. Let Σ be a finite alphabet and Σ^* be the set of finite words built on Σ. For a binary predicate $B \subseteq \Sigma^* \times \Sigma^*$, we write $B(x)$ for the set of y such that $B(x, y)$ holds. To simplify the discussion, we consider predicates B such that $B(x)$ is finite for all x.

The *enumeration problem* Π_B is the function that associates $B(x)$ to x. Intuitively, it lists all solutions y to problem instance x.

EnumP. EnumP is the class of enumeration problems Π_B such that $B \in \mathsf{P}$, *i.e.*, it is in PTIME to check whether $B(x, y)$ holds (whether y is a solution to x). It is the equivalent of NP in enumeration.

An EnumP-*complete problem* is a problem in EnumP to which all problems in EnumP can be reduced by parsimonious reductions. A (weak) *parsimonious reduction* R from a problem A to problem B is a PTIME transformation from instances of A to instances of B such that for any instance x of A, the number of solutions to x is equal to the number of solutions to instance $R(x)$ of problem B [85].

Theorem 1: SubIso *is* EnumP-*complete.* □

Proof sketch: We show the EnumP-completeness of SubIso by parsimonious reduction from

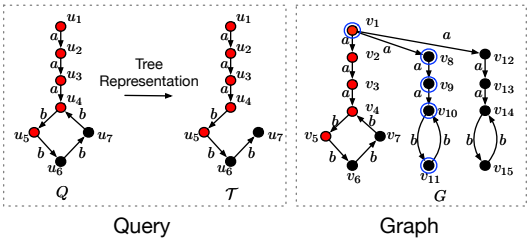


Fig. 1. Given a graph G and a pattern Q, VF3$_M$ first builds a tree representation $\mathcal{T}$ of Q based on a matching order O, and then enumerates matches in $Q(G)$ using $\mathcal{T}$ and order O.

Π_{SAT}, which enumerates all satisfying assignments of a 3CNF formula and is known to be EnumP-complete [105]. Given a 3SAT formula $\Phi = C_1 \wedge \cdots \wedge C_n$, the reduction constructs a graph G with seven vertices for each clause C_i ($i \in [1, n]$), and define a pattern Q as a n-clique. □

Remark. Parsimonious reductions are classified as weak or strong. The weak form preserves the number of solutions under a PTIME transformation, while the strong form establishes a PTIME bijection between individual solutions. We adopt the weak notion, which is standard in counting complexity [85], whereas the strong form is mainly used for approximate counting problems [46].

OutputP. A problem $\Pi_B \in$ EnumP is in OutputP if for all instances x of Π_B, the time for computing all solutions in $B(x)$ is bounded by a polynomial in the size $|x|$ of input and the size $|B(x)|$ of the output. Such a problem is said to be *output polynomial* [97].

Intuitively, Π_B is in OutputP if it can be solved in PTIME in the sizes of the input and output. The size of output is taken into account since the number of solutions may be exponential in the size of the input, *e.g.,* SubIso. The cost is measured as the total time needed to compute all solutions in $B(x)$ to input x, using a RAM machine.

Abusing the notation, we say that an enumeration algorithm for Π_B is *output polynomial* if it takes PTIME in $|x|$ and $|B(x)|$.

Remark. We adopt the TotalP class (*i.e.,* OutputP) [98] for problems solvable in output-polynomial time, *i.e.,* enumerating all complete solutions in PTIME in input and output size. If a randomized algorithm may omit some solutions, the problem lies in TotalBPP [98].

3 Programming with ML Oracles

This section develops revisions VF3$_M$ of algorithm VF3 [28] by incorporating an ML oracle $\mathcal{M}$. We first review VF3 (Section 3.1) and present a template for VF3$_M$ (Section 3.2). Based on the template, we then develop three VF3$_M$ algorithms (Section 3.3).

3.1 Algorithm VF3

We first review VF3 [28, 29], a popular exact algorithm for SubIso. Given a pattern Q and a graph G, it employs a recursive procedure Enumerate to find all full matches of Q in G, and reduces the search space by using efficient heuristic rules. Here we improve the original algorithm VF3 [28, 29] by incorporating existing optimization strategies that have been verified effective for SubIso [128].

Algorithm. Given Q and G, VF3 computes the set $Q(G)$ of matches of Q in G. It first sets a matching order O of Q using the Maximum Likelihood Estimation method. Here a *matching order* is a permutation of the pattern vertices V_Q in Q. Then it assigns each vertex in G or Q to a class such that vertices from the same class can be matched. Intuitively, it computes a set $C(u)$ of candidates matches from G for each pattern vertex u in Q. Then, it constructs a tree representation $\mathcal{T}$ of Q

Input: A graph G, a pattern Q, a matching order O, a classification C, a tree representation $\mathcal{T}$ of Q, and the current partial match ρ.
Output: All full matches of Q in G that extend ρ.

```
1.  if IsGoal(ρ) then
2.      return {ρ};
3.  if IsDead(ρ) then
4.      return ∅;
5.  (u_c, v_c) := (⊥, ⊥);
6.  (u_n, v_n) := GetCan(G, Q, O, C, ρ, T, (u_c, v_c));
7.  while (u_n, v_n) ≠ (⊥, ⊥) do
8.      if IsFeasible(G, Q, ρ, u_n, v_n) then
9.          ρ_c := ρ + (u_n, v_n);
10.         ΔOutput := Enumerate(G, Q, O, C, T, ρ_c);
11.         Output := Output ∪ ΔOutput;
12.     (u_n, v_n) := GetCan(G, Q, O, C, ρ, T, (u_n, v_n));
13. return Output;
```

Fig. 2. Procedure Enumerate

based on O, *i.e.,* a vertex u_p is a parent of u in $\mathcal{T}$ if u_p has an edge connected to u and appears before u in O. It also builds ρ_0, the initial partial match $\emptyset$. After these, it enumerates matches in $Q(G)$ following the order O via procedure Enumerate.

Procedure Enumerate. As shown in Figure 2, given graph G, pattern Q, candidates C for pattern vertices, a tree representation $\mathcal{T}$ of Q and a partial match ρ, Enumerate recursively extends the partial match ρ to full matches, if possible, following the order O.

Enumerate works as follows. It first checks whether ρ is already a full match; if so, it returns ρ as part of Output (lines 1-2). Otherwise, it checks whether ρ is invalid, *e.g.,* a pattern vertex in ρ has more neighbors than its candidate matches in ρ; if so, it returns $\emptyset$, *i.e.,* the current partial match cannot be extended to a full match (lines 3-4).

Otherwise, Enumerate iteratively extends a partial match ρ by mapping the next pattern vertex u_n to candidates $v \in C(u_n)$ (lines 5–12). It selects $v_n \in C(u_n)$ that matches u_n (lines 5–6) such that v_n and u_n belong to the same class and have isomorphic connections with vertices in ρ; *i.e.,* if (u_n, l, u_1) (resp. (u_1, l, u_n)) exists in pattern Q, then $(v_n, l, \rho(u_1))$ (resp. $(\rho(u_1), l, v_n)$) is in graph G. Function GetCan selects the next candidate following order O. When $(u_n, v_n) = (\perp, \perp)$, the list ends and the loop terminates (line 7).

Expanding ρ with a candidate (u_n, v_n) from GetCan, denoted $\rho + (u_n, v_n)$, Enumerate checks whether it forms a bijective mapping. It then calls the Boolean function IsFeasible, which inspects the 2-hop neighbors of u_n and v_n using heuristic rules to decide whether further extension is possible (line 8). If so, Enumerate recursively expands $\rho + (u_n, v_n)$ (lines 9–11). After processing v_n, it continues with other candidates in $C(u_n)$ until all are examined (lines 7, 12).

When processing $\rho + (u_n, v_n)$, Enumerate may recurse deeper by expanding to $(u_{n+1}, v_{n+1}), \ldots, (u_{n+m}, v_{n+m})$, only to find that the branch cannot yield a full match (line. 3). At this point, it backtracks to another candidate v'_n for u_n. Such backtracking is costly when m is large, as all work spent expanding $\rho + (u_n, v_n)$ is wasted.

Remark. Function GetCan selects the next candidate match (u_{n+1}, v_{n+1}) following matching order O: (a) if $(u_n, v_n) = (\perp, \perp)$ and $\rho = \emptyset$, it picks the first pattern vertex in O and its first candidate in $C(u_n)$ (line 6 of Enumerate); (b) if $(u_n, v_n) = (\perp, \perp)$ but $\rho \neq \emptyset$, it retrieves the next pattern vertex u_{n+1} in O based on ρ and selects its first candidate; (c) if $(u_n, v_n) \neq (\perp, \perp)$ and $\rho \neq \emptyset$, it retrieves the next vertex v_{n+1} in $C(u_n)$ (line 12); and (d) when all pattern vertices and candidates are processed, *i.e., u_n* is the last in O and v_n is the last in $C(u_n)$, GetCan returns $(\perp, \perp)$ to terminate (line 12).

Input: A graph G, a pattern Q, a partial match ρ, a vertex u in Q and a vertex v in G.
Output: Whether there exists a full match extended from ρ.
1. **if** $\mathcal{M}(G, Q, \rho + (u_n, v_n)) \geq \delta_{\mathsf{T}}$ **then**
2. **return** ReduceFP$(G, Q, \mathcal{O}, \mathcal{T}, \rho)$;
3. **return** ReduceFN$(G, Q, \mathcal{O}, \mathcal{T}, \rho)$;

Fig. 3. Procedure IsFeasible$_{\mathsf{M}}$

Optimizations. The VF3 algorithm of Figure 2 extends the original algorithm [28, 29] with the following optimization strategies.

Classification. We adopt graded simulation [84] to compute the candidate set $C(u)$. Weaker than subgraph isomorphism, graded simulation pre-filters vertices in G that cannot match u. Given a graph G and a pattern Q, it computes the maximum binary relation $\mathcal{G} \subseteq V_Q \times V_G$ such that $(u, v) \in \mathcal{G}$ iff the one-hop neighbors of u can be mapped to those of v. Specifically, $(u, v) \in \mathcal{G}$ when: (1) u and v share the same label; (2) for u's m distinct outgoing edges (u, l_i, u_i), v has corresponding distinct outgoing edges (v, l_i, v_i) with $(u_i, v_i) \in \mathcal{G}$; and (3) the same holds for incoming edges. The relation $\mathcal{G}$ can be computed in $O(|V|(|V| + |E|)|V_Q|(|V_Q| + |E_Q|))$ time [82].

We include in $C(u)$ all such vertices v in G that $(u, v) \in \mathcal{G}$.

Matching order. We adopt a degree-based strategy for determining the matching order $\mathcal{O}$ [23], starting from the vertex with the largest degree and iteratively selecting the one with the most backward neighbors. As shown in a recent survey [128], this strategy performs well for SubIso, though other methods such as topological ordering [23] and reinforcement learning [110] can also be used.

Example 2: Continuing with Example 1, VF3 finds matches of Q in G as follows. (1) It first sets a matching order $\mathcal{O}$ of Q, *e.g.,* $u_1, \ldots, u_7$, represented by tree $\mathcal{T}$ in Figure 1. (2) Following $\mathcal{O}$, it recursively enumerates matches in $Q(G)$, mapping u_1 to v_1 first, followed by finding matches of $u_2, \ldots, u_7$. It returns the unique full match. □

3.2 Algorithm Template for VF3$_M$

We next outline a template VF3$_M$ for revising the VF3 algorithm, by incorporating an ML model $\mathcal{M}$ into VF3 as an oracle.

ML oracle $\mathcal{M}$. As will be seen in Section 5, we will train an ML model $\mathcal{M}$ that, given a graph G, a pattern Q and a partial match ρ, returns a numeric value in $[0, 1]$ indicating the confidence of the prediction for ρ to be extended to a full match. We set a configurable threshold δ_{T} such that $\mathcal{M}(G, Q, \rho) \geq \delta_{\mathsf{T}}$ if the prediction of $\mathcal{M}$ is highly confident to be true, *i.e.,* ρ is valid. Departing from procedure IsFeasible in Enumerate that only inspects the 2-hop neighbors of pattern vertices, $\mathcal{M}$ learns topological connections of graphs and patterns, and *directly* predicts whether ρ is valid or not.

Intuitively, when the ML model $\mathcal{M}$ is highly accurate, one can use $\mathcal{M}$ instead of IsFeasible. After expanding ρ with (u, v), VF3$_M$ first calls $\mathcal{M}$ to predict whether $\rho + (u, v)$ can be extended to a full match; if $\mathcal{M}$ returns true, then it extends ρ with (u, v) just like VF3; otherwise it does not extend ρ with (u, v), without backtracking.

Example 3: Consider graph G and pattern Q in Figure 1. There exists only one match of Q in G, *i.e.,* the leftmost part of G. Note that the two cycles with two vertices in G cannot match the cycle with four vertices in Q. Such mismatches cannot be filtered out by graded simulation [84] in node classification described earlier.

One can verify the following. (1) When VF3 is invoked to compute $Q(G)$, procedure Enumerate is called 16 times, one for each vertex in G, plus the first call by VF3. (2) In contrast, if we replace

Input: $G, Q, \rho, u \in Q$, and $v \in G$ as in Figure 3.
Output: true if ρ is valid.
1. VF3_check := IsFeasible(G, Q, ρ, u_n, v_n);
2. **if** $\mathcal{M}(G, Q, \rho + (u_n, v_n)) \geq \delta_{\mathsf{T}}$ and VF3_check **then**
3. **return** true;
4. **if** $\mathcal{M}(G, Q, \rho + (u_n, v_n)) < \delta_F$ **then**
5. **return** false;
6. **return** VF3_check;

Fig. 4. Procedure IsFeasible$_{\mathsf{M}}$ in VF3$^{O}_{M}$

IsFeasible with $\mathcal{M}$ and if $\mathcal{M}$ predictions are accurate, we only need to invoke Enumerate 8 times, which finds the unique match of Q in G. Indeed, when $\mathcal{M}$ is accurate, it correctly predicts that the partial match mapping u_2 to v_8 or v_{12} cannot lead to a full match, and thus does not call procedure Enumerate for further inspection. □

Template VF3$_M$. However, when $\mathcal{M}$ is not perfectly accurate, it may incorrectly predict false on a true match and miss the match (false negative, FN). It may also predict true on false matches (false positive, FP) and incurs redundant checking. To reduce the impact of FNs and FPs, we develop VF3$_M$ by including two procedures.

Template VF3$_M$ is a mild variation of VF3 with the following. (1) It replaces IsFeasible (line 8 of Figure 2) with IsFeasible$_{\mathsf{M}}$ given in Figure 3. It employs $\mathcal{M}$ to predict whether ρ is valid after expanding ρ with (u, v). (2) To reduce FPs of $\mathcal{M}$ predictions, when $\mathcal{M}$ returns true, it may call Boolean function ReduceFP to conduct further checking (line 2). (3) To reduce FNs of $\mathcal{M}$, when $\mathcal{M}$ predicts false, it may call Boolean function ReduceFN to mitigate bad predictions (line 3). ReduceFP and ReduceFN serve as *backup plans* for bad predictions of $\mathcal{M}$, and will be elaborated in Section 3.3.

3.3 Three VF3$_M$ Algorithms

Based on the template, we next develop three VF3$_M$ algorithms with different strategies for reducing FPs and FNs of $\mathcal{M}$ predictions.

(1) Do nothing. The first VF3$_M$ algorithm, denoted by VF3$^{N}_{M}$, opts to completely trust the ML model $\mathcal{M}$. It follows the predictions of $\mathcal{M}$. Here ReduceFP (resp. ReduceFN) simply returns true (resp. false) when $\mathcal{M}$ predicts true (resp. false). In other words, VF3$^{N}_{M}$ simply replaces IsFeasible in line 8 of Figure 2 with $\mathcal{M}(G, Q, \rho) \geq \delta_{\mathsf{T}}$ for a predefined threshold δ_{T}, indicating that $\mathcal{M}$ predicts true; the rest of the VF3 code of Figure 2 remains unchanged.

As will be seen in Section 4, VF3$^{N}_{M}$ is (a) consistent, and (b) output-polynomial with a high probability. However, it may miss true matches when $\mathcal{M}$ makes FN predictions, *i.e.,* the recall of VF3$^{N}_{M}$ is impacted by bad $\mathcal{M}$ predictions, where recall is the ratio of matches returned by VF3$^{N}_{M}$ to all matches in $Q(G)$.

(2) VF3 **as a backup**. We develop VF3$^{O}_{M}$ to improve the recall of VF3$^{N}_{M}$. We set another configurable threshold δ_{F} such that $\mathcal{M}(G, Q, \rho) < \delta_{\mathsf{F}}$ if the prediction of $\mathcal{M}$ is highly confident to be false, *i.e., ρ* cannot be extended to a full match. Note that $\delta_{\mathsf{F}} < \delta_{\mathsf{T}}$.

Algorithm VF3$^{O}_{M}$ replaces IsFeasible of VF3 with Boolean function IsFeasible$_M$ shown in Figure 4. It makes decisions as follows. (a) It returns true if both $\mathcal{M}$ predicts true (*i.e.,* the confidence of $\mathcal{M}$ is at least δ_{T}) and IsFeasible returns true, to reduce FPs (line 2-3). Here IsFeasible serves as ReduceFP. (b) If $\mathcal{M}(G, Q, \rho) < \delta_{\mathsf{F}}$, *i.e.,* if $\mathcal{M}$ is highly confident to predict false, it follows the prediction and returns false (lines 4-5). (c) Otherwise, *i.e.,* when $\mathcal{M}$ is confident enough to predict neither true nor false, it resorts to IsFeasible to check the two-hop neighbors of u and v as in VF3. That is, IsFeasible is invoked as ReduceFN to improve the recall (line 6).

Input: G, Q, ρ, $u \in Q$, and $v \in G$ as in Figure 3.
Output: true if ρ is valid.
1. VF3_check := IsFeasible(G, Q, ρ, u_n, v_n);
2. **if** $\mathcal{M}(G, Q, \rho + (u_n, v_n)) \geq \delta_T$ and VF3_check **then**
3. **return** true;
4. **if** $\mathcal{M}(G, Q, \rho + (u_n, v_n)) < \delta_F$ **then**
5. **return** DeepCheck(G, Q, ρ, u_n, v_n);
6. **return** VF3_check;

Fig. 5. Procedure IsFeasible$_M$ in VF3^D_M

Input: G, Q, ρ, $u \in Q$, and $v \in G$ as in Figure 3.
Output: true if ρ is valid.
1. let $u'_1, \ldots, u'_{k_\rho}$ be the remaining pattern vertices to be matched; $j := 0$;
2. **for each** $j \leq S(k_\rho)$ **then**
3. randomly sample one vertex v_i in $C(u'_i)$ for each $i \in [1, k_\rho]$;
4. $\rho_c := \rho + (u'_1, v_1) + \cdots + (u'_{k_\rho}, v_{k_\rho})$; $j := j + 1$;
5. **if** ρ_c is a full match **then**
6. **return** true;
7. **return** false;

Fig. 6. Procedure DeepCheck in VF3^D_M

We will see that the recall of VF3^O_M is 1 with a high probability. Moreover, it is 1-robust with a high probability, and guarantees to be consistent. As a price to pay, it is unlikely output-polynomial.

(3) Deep checking. When $\mathcal{M}$ makes FN predictions, VF3^O_M may miss valid partial matches and hamper its recall. To reduce FNs, we develop VF3^D_M that replaces IsFeasible with IsFeasible$_M$ of Figure 5. If $\mathcal{M}$ predicts false, it conducts deep checking (line 2) as ReduceFN.

Deep checking works as follows. Given partial match ρ, let $u'_1, \ldots, u'_{k_\rho}$ be the remaining vertices of pattern Q to be matched, in the matching order O. Here k_ρ=$|V_Q|-|\rho|$ and $|\rho|$ is the number of pattern vertices in ρ. Then deep checking first samples a vertex v_1 from candidate set $C(u'_1)$ such that v_1 and u'_1 have "isomorphic" connections with vertices in ρ, and then expands ρ with (u'_1, v_1). It repeats the step to sample one vertex v_i for each u'_i $(i \in [2, k_\rho])$ following O, and generates $\rho_c = \rho + (u'_1, v_1) + \cdots + (u'_{k_\rho}, v_{k_\rho})$. It checks whether ρ_c makes a full match and returns true if so.

To reduce FNs, deep checking is conducted for $S(k_\rho)$ times, where $S(k_\rho) = |G|^{k_\rho/|Q|} \leq |G|$, following a sampling rate decay strategy [107]. Intuitively, the more remaining pattern vertices are (*i.e.,* the larger k_ρ is), the more deep checking steps are performed. This strategy is to strike a balance between the complexity and accuracy. Other strategies can be used instead to reduce FNs.

As shown in Figure 6, DeepCheck generates at most $S(k_\rho)$ extensions of ρ (line 2). Each extension samples one vertex v_i in $C(u'_i)$ for each $i \in [1, k_\rho]$ (line 3), and forms $\rho_c = \rho + (u'_1, v_1), \ldots, (u'_{k_\rho}, v_{k_\rho})$ (line 4). DeepCheck checks whether ρ_c is valid; if so, it returns true. It returns false if none of the $S(k_\rho)$ extensions finds a full match.

Example 4: Consider graph G, pattern Q and matching order O (tree $\mathcal{T}$) in Figure 7. Pattern Q has a unique full match in G (blue circles). VF3^D_M first selects candidate vertices for each pattern vertex via graded simulation, then enumerates matches in $Q(G)$ following O. Unlike VF3, after mapping u_2 to v_2, VF3^D_M calls $\mathcal{M}$ to predict whether partial match ρ is valid. If $\mathcal{M}$ returns false and $\mathcal{S}(k_\rho)$=2, DeepCheck explores vertices $u_3, \ldots, u_5$ of Q along paths from two children, *e.g.,* v_3 and v_5, of v_2 in G, and returns true on the path from v_5, overriding $\mathcal{M}$'s prediction and reducing FNs. □

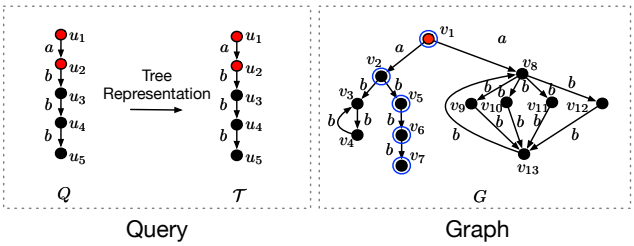


Fig. 7. An example to show how $\mathsf{VF3}^D_M$ runs.

	$\mathsf{VF3}^N_M$	$\mathsf{VF3}^O_M$	$\mathsf{VF3}^D_M$
Output polynomial	with high probability (Theorem 6)	× (Theorem 2)	× (Theorem 2)
Competitive ratio	$2\|G\|Cf(Q,\mathcal{M})$ (Corollary 7)	$5N_c(f(Q,\mathcal{M})+\|G\|\|Q\|)$ (Theorem 8)	$5N_c(f(Q,\mathcal{M})+2\|G\|^2\|Q\|)$ (Theorem 8)
Consistency	✓ (Theorem 3)	✓ (Theorem 3)	✓ (Theorem 3)
Robustness	β-robust (Theorem 4)	β-robust (Theorem 5)	β-robust (Theorem 5)

Table 1. Positive result summary

4 Performance Guarantees

This section studies the efficiency and accuracy of $\mathsf{VF3}_M$ algorithms. We report negative (Section 4.1) and positive (Section 4.2) results.

Below we first introduce evaluation measures for the proposed methods that integrate algorithmic techniques with ML predictions.

Efficiency. We adopt the following two measures for efficiency.

Output polynomial. As remarked in Section 2, we say that a $\mathsf{VF3}_M$ algorithm is *output-polynomial* for SubIso if its time cost can be expressed as a polynomial in the size of input (graph G and pattern Q) and the size of output (the set of matches $|Q(G)|$).

Competitive ratio. Following Lykouris and Vassilvitskii [73], we define the *competitive ratio* λ of a $\mathsf{VF3}_M$ algorithm with respect to a yardstick optimal algorithm such that

$$\text{cost}(Q,G) \leq \lambda\ \mathsf{OPT}(Q,G),$$

where $\text{cost}(Q,G)$ denotes the cost of $\mathsf{VF3}_M$ for computing $Q(G)$, and OPT is the cost incurred by the optimal offline algorithm. The smaller the ratio λ is ($\lambda \geq 1$), the more efficient the algorithm is.

Accuracy. Consider a $\mathsf{VF3}_M$ algorithm $\mathcal{A}$. We measure the accuracy of $\mathcal{A}$ in terms of F1 measure: $F1(\mathcal{A}) = \frac{2\cdot\text{precision}\cdot\text{recall}}{(\text{precision}+\text{recall})}$. Here the *precision* of $\mathcal{A}$ is the ratio of true matches found by $\mathcal{A}$ to all matches returned by $\mathcal{A}$, *i.e., precision* = $\frac{|\text{Output}\cap Q(G)|}{|\text{Output}|}$, and the *recall* is the ratio of true matches returned by $\mathcal{A}$ to all true matches in $Q(G)$, *i.e., recall* = $\frac{|\text{Output}\cap Q(G)|}{|Q(G)|}$, for patterns Q and graphs G.

To evaluate the impact of the embedded ML model $\mathcal{M}$ on the accuracy of $\mathcal{A}$, we also study the following two measures.

Consistency. Denote the error rate of model $\mathcal{M}$ by η, which is the probability that $\mathcal{M}$ makes wrong predictions. A $\mathsf{VF3}_M$ algorithm $\mathcal{A}$ with $\mathcal{M}$ is *consistent* if $F1(\mathcal{A}) = 1$ when $\eta = 0$, *i.e.,* when $\mathcal{M}$ tends to be error-free, $\mathcal{A}$ finds all and only the matches in $Q(G)$.

Robustness. A $\mathsf{VF3}_M$ algorithm $\mathcal{A}$ with ML model $\mathcal{M}$ is *β-robust* if $F1(\mathcal{A}) \geq \beta$ for a bound β regardless of η; *i.e.,* it is still able to find true matches even when $\mathcal{M}$ makes arbitrarily bad predictions.

4.1 Negative Results

We start with an impossibility result.

Theorem 2: *No algorithm exists for* SubIso *that is both output polynomial and β-robust for any positive constant β unless* fewP = P. □

Here fewP denotes the class of problems recognizable by a nondeterministic Turing machine in polynomial time with only polynomially many solutions [9]. It is known that P ⊆ fewP [56, 85], and that fewP = P is considered as unlikely as P = NP [37].

Proof sketch: Assume by contradiction that such an algorithm $\mathcal{A}$ exists with $F1(\mathcal{A}) \geq \beta$; we show that fewP = P. (1) We first construct a parsimonious reduction from fewP to a subset of SubIso, which consists of instances (Q, G) such that $Q(G)$ contains only polynomially many matches of Q in G [34, 83]. (2) Using $\mathcal{A}$, we design a PTIME algorithm $\mathcal{B}$ to check whether T accepts x. Hence, fewP⊆P and as P ⊆ fewP [56, 85], it follows that fewP = P. □

Remark. Theorem 2 does not follow from the NP-hardness of SubIso. Rather, it targets EnumP, the class of enumeration problems with PTIME verification, not decision problems in NP. It shows that for EnumP-complete problems such as SubIso, no algorithm can be both output-polynomial and β-robust for any constant β, thereby ruling out attempts to achieve both properties simultaneously.

The result does not extend to all NP-complete problems. For example, the edge dominating set problem is NP-complete [45], yet its enumeration is output-polynomial, *i.e.,* all minimal edge dominating sets can be listed in polynomial time in the sizes of both input and output [47, 92]. Moreover, its enumeration is also 1-robust (*i.e.,* $\beta = 1$), since it is an exact algorithm and returns all minimal edge dominating sets [47]. Thus, for such problems, both properties are simultaneously attainable, as opposed to SubIso.

Hence, Theorem 2 delineates a boundary for achieving accuracy and efficiency together, which is previously unknown.

4.2 Positive Results

Despite Theorem 2, we show that $\mathsf{VF3}^N_M$, $\mathsf{VF3}^O_M$ and $\mathsf{VF3}^D_M$ achieve provable performance guarantees. Table 1 summarizes these positive results, outlining the conditions, output-polynomial bounds and competitive ratios of the three algorithms. These findings provide insights into developing practical algorithms for EnumP-complete problems, and indicate that $\mathsf{VF3}_M$ algorithms become increasingly effective when ML model accuracy improves.

4.2.1 Accuracy. We start with precision and consistency.

Consistency. All the three $\mathsf{VF3}_M$ algorithms are consistent.

Theorem 3: *Algorithms* $\mathsf{VF3}^N_M$, $\mathsf{VF3}^O_M$ *and* $\mathsf{VF3}^D_M$ *(1) have* precision=1 *and (2) are consistent.* □

Proof sketch: (1) For precision, note that every match returned by the three $\mathsf{VF3}_M$ algorithms belongs to $Q(G)$, since a match is output only when it is full and valid (lines 1–2 of Enumerate).

(2) When the error rate η of $\mathcal{M}$ is 0, the F1 score of all three algorithms equals 1. Specifically, $\mathsf{VF3}^N_M$ achieves recall 1, as every full match $\rho \in Q(G)$ is returned when $\eta = 0$. Since all three maintain precision 1 and the recalls of $\mathsf{VF3}^O_M$ and $\mathsf{VF3}^D_M$ are no lower than that of $\mathsf{VF3}^N_M$, the result follows. □

Robustness. We have a lower bound for the robustness of $\mathsf{VF3}^N_M$ using its output. Here we assume *w.l.o.g.* that the prediction of $\mathcal{M}$ is "monotonically decreasing" (see Section 5), *i.e.,* if a partial match ρ is extended from partial one ρ', when $\mathcal{M}(G, Q, \rho') =$ false, the prediction $\mathcal{M}(G, Q, \rho)$ must also be false. Intuitively, since ρ is extended from ρ', the search space for full matches of ρ' is larger, and when ρ' cannot be extended to a full match, neither can ρ.

Theorem 4: *Given graph G and pattern Q, for constant $\varepsilon \in (0,1)$, $\mathsf{VF3}^N_M$ is $\frac{2\mathsf{Output}_T}{2\mathsf{Output}_T+f(\varepsilon)}$-robust with a probability of at least $1-\varepsilon$.* □

Here (a) Output_T denotes the number of full matches returned by $\mathsf{VF3}^N_M$; and (b) $f(\varepsilon) = N_{\mathcal{N}}\mathcal{B}$, where $N_{\mathcal{N}}$ is the number of false predictions of $\mathcal{M}$, and $\mathcal{B} = \frac{1+\eta|G|-|G|+\sqrt{(|G|-1-|G|\eta)^2-4(1-\frac{\varepsilon}{N_{\mathcal{N}}})}}{2\eta}$.

Proof sketch: Since $\mathsf{VF3}^N_M$ has precision 1, it suffices to show that its recall is at least $\frac{\mathsf{Output}T}{\mathsf{Output}T+N_{\mathcal{N}}\mathcal{B}}$. We first prove that, for each false prediction of $\mathcal{M}$, the number of missed matches is bounded by $\mathcal{B}$ with probability at least $1-\frac{\varepsilon}{N_{\mathcal{N}}}$. Then, by applying the generalized Bonferroni inequality [30], we establish that the recall of $\mathsf{VF3}^N_M$ is at least $\frac{\mathsf{Output}_T}{\mathsf{Output}T+N_{\mathcal{N}}\mathcal{B}}$ with probability at least $1-\varepsilon$. □

We next show that $\mathsf{VF3}^O_M$ and $\mathsf{VF3}^D_M$ are "almost" 1-robust. As commonly found in practice, $\mathcal{M}$ makes wrong predictions typically when $\mathcal{M}(G,Q,\rho)$ falls in the range $[\delta_F, \delta_T)$, *i.e.,* when $\mathcal{M}$ is neither confident to predict true nor certain to predict false. To evaluate the impact of range $[\delta_F, \delta_T)$ on $\mathsf{VF3}_M$, denote by p_{u} the ratio of full matches at which the confidences of $\mathcal{M}(G,Q,\rho)$ are in the range $[\delta_F, \delta_{\mathsf{T}})$ to all full matches missed by $\mathsf{VF3}_M$. Denote by Output_T (resp. Output_{FT} and Output_F) the number of returned matches at which $\mathcal{M}(G,Q,\rho)$ is in the range $[\delta_T, 1]$ (resp. $[\delta_F, \delta_{\mathsf{T}})$ and $[0, \delta_F)$).

Theorem 5: *With a probability of at least $1-\varepsilon$, algorithm $\mathsf{VF3}^O_M$ (resp. $\mathsf{VF3}^D_M$) is $\frac{2(\mathsf{Output}_T+\mathsf{Output}_{FT})}{2\mathsf{Output}_T+2\mathsf{Output}_{FT}+(1-p_{\mathsf{u}})N_{\mathcal{N}}\mathcal{B}}$-robust (resp. $\frac{2(\mathsf{Output}_T+\mathsf{Output}_{FT}+\mathsf{Output}_F)}{2\mathsf{Output}_T+2\mathsf{Output}_{FT}+2\mathsf{Output}_F+(1-p_{\mathsf{u}})N_{\mathcal{N}}\mathcal{B}}$-robust).* □

Observe that when the ratio p_{u} approximates 1, *i.e.,* when full matches missed by $\mathsf{VF3}_M$ fall in $[\delta_F, \delta_{\mathsf{T}})$, the recalls of $\mathsf{VF3}^O_M$ and $\mathsf{VF3}^D_M$ approach 1 and both algorithms are almost 1-robust.

Proof sketch: We first show that the recall of $\mathsf{VF3}^O_M$ is at least $\frac{\mathsf{Output}T+\mathsf{Output}FT}{\mathsf{Output}T+\mathsf{Output}FT+(1-p_u)N_{\mathcal{N}}\mathcal{B}}$. By Theorem 4, $\mathsf{VF3}^N_M$ misses at most $N_{\mathcal{N}}\mathcal{B}$ full matches with probability $1-\varepsilon$. $\mathsf{VF3}^O_M$ improves recall by additionally returning full matches whose confidence scores fall within $[\delta_{\mathsf{F}}, \delta_{\mathsf{T}})$. At most $(1-p_u)N_{\mathcal{N}}\mathcal{B}$ full matches have confidences below δ_{F}, hence the recall of $\mathsf{VF3}^O_M$ satisfies the bound above. The analysis for $\mathsf{VF3}^D_M$ follows analogously. □

4.2.2 Efficiency. Below we first study when the $\mathsf{VF3}_M$ algorithms are output polynomial. We then investigate their competitive ratios.

Output polynomial. $\mathsf{VF3}^N_M$ is "almost" output polynomial.

Theorem 6: *Given graph G and graph pattern Q, $\mathsf{VF3}^N_M$ is output polynomial with a high probability. For a constant $\varepsilon \in (0,1)$, $\mathsf{VF3}^N_M$ runs in $|Q||G|(\mathsf{Output}_T+1)\times C$ time with a probability of at least $1-\varepsilon$, where $C = \frac{1+(1-\eta)|G|-|G|+\sqrt{(|G|-1-|G|(1-\eta))^2-4(1-\varepsilon)}}{2(1-\eta)}$.* □

Proof sketch: (1) When $\mathcal{M}$ is error-free, it suffices to show that Enumerate is invoked at most $O(|G|(\mathsf{Output}_T+1))$ times. (a) If $\mathcal{M}$ predicts true for a partial match ρ, then ρ is valid, and the number of such true predictions is bounded by $O(|Q|\mathsf{Output}_T)$. (b) If $\mathcal{M}$ predicts false, $\mathsf{IsFeasible}_{\mathsf{M}}$ immediately returns false in $O(1)$ time.

(2) We next show that when the model error rate is η, $\mathsf{VF3}^N_M$ remains output-polynomial with high probability. Consider four types of predictions made by $\mathcal{M}$: (a) true negatives (TN), (b) false negatives (FN), (c) true positives (TP), and (d) false positives (FP). Cases (a)–(c) follow the same analysis as in (1). For case (d), we show that, with high probability, each minimal partial match ρ'_k yields at most C' false-positive predictions when $\mathcal{M}$ evaluates its extensions, where C' generalizes C. Applying the generalized Bonferroni inequality [30], we further bound the total number of false positives by $|G|(\mathsf{Output}_T+1)C$ with probability at least $1-\varepsilon$. □

Remark. (1) By Theorem 2, it is beyond reach in practice to train an error-free ML model $\mathcal{M}$ for SubIso, since otherwise, $\mathsf{VF3}^N_M$ is both 1-robust and output polynomial. This said, more accurate ML models for SubIso yield a higher recall for algorithm $\mathsf{VF3}^N_M$. We will train an ML model for SubIso with high accuracy in Section 5.

(2) In contrast, when $p_{\mathsf{u}} = 1$, *i.e.,* when all those full matches missed by $\mathcal{M}$ fall in the confidence range $[\delta_F, \delta_{\mathsf{T}})$, $\mathsf{VF3}^O_M$ and $\mathsf{VF3}^D_M$ call IsFeasible as VF3, and they behave the same as VF3. Hence unless $\mathcal{M}$ is accurate, they cannot be output-polynomial by Theorem 2.

Competitive ratio. We start with algorithm $\mathsf{VF3}^N_M$.

Corollary 7: *Given graph G and pattern Q, for a constant $\varepsilon \in (0,1)$, the competitive ratio of $\mathsf{VF3}^N_M$ is $2|G|Cf(Q,\mathcal{M})$ with probability of at least $1-\varepsilon$, where $C = \frac{1+(1-\eta)|G|-|G|+\sqrt{(|G|-1-|G|(1-\eta))^2-4(1-\varepsilon)}}{2(1-\eta)}$, and $f(Q,\mathcal{M})$ denotes the cost for $\mathcal{M}$ to make a prediction.* □

Proof sketch: By Theorem 6, the cost of $\mathsf{VF3}^N_M$ is bounded by $|G|(\mathsf{Output}_T + 1)Cf(Q,\mathcal{M})$ with high probability. Since $\mathsf{VF3}^N_M$ has precision 1, there exist at least Output_T full matches in $Q(G)$; hence $\mathsf{Output}_T \leq \mathsf{OPT}$, where OPT is the optimal offline cost. Thus, $|G|(\mathsf{Output}_T + 1)Cf(Q,\mathcal{M}) \leq 2|G|Cf(Q,\mathcal{M}) \cdot \mathsf{OPT}$. In practice, $f(Q,\mathcal{M})$ is typically polynomial once $\mathcal{M}$ is trained. □

We next study the competitive ratio of $\mathsf{VF3}^O_M$ and $\mathsf{VF3}^D_M$. Recall p_{u} given earlier. Let $N_c = |G|C|Q|p_{\mathsf{u}}\frac{N_N}{\mathsf{Output}_T}\mathcal{B}$.

Theorem 8: *Given graph G and pattern Q, for constant $\varepsilon \in (0,1)$, competitive ratio of $\mathsf{VF3}^O_M$ (resp. $\mathsf{VF3}^D_M$) is $5N_c(f(Q,\mathcal{M})+|G||Q|)$ (resp. $5N_c(f(Q,\mathcal{M})+2|G|^2|Q|)$) with probability at least $1-\varepsilon$.* □

Proof sketch: We establish the competitive ratio by partitioning partial matches according to $\mathcal{M}$'s confidence: (a) above δ_{T}, (b) between δ_{T} and δ_{F}, and (c) below δ_{F}. We show that the numbers of full matches in these parts are bounded by $\mathsf{Output}T$, $p_{\mathsf{u}}N_N\mathcal{B}$, and $|G|(\mathsf{Output}_T + 1)$, respectively, from which the runtime bounds of $\mathsf{VF3}^O_M$ and $\mathsf{VF3}^D_M$ follow. □

5 Embedding and Learning

This section trains an ML model $\mathcal{M}$ for $\mathsf{VF3}_M$. Given a pattern Q, a graph G and a partial match ρ of Q in G, model $\mathcal{M}$ predicts whether ρ is valid, *i.e.,* whether it can be extended to a full match of Q in G.

Challenges. Training $\mathcal{M}$ is nontrivial. Model $\mathcal{M}$ must (a) be efficient, as $\mathsf{VF3}_M$ may invoke it exponentially many times; (b) be accurate to preserve recall; (c) encode both edge labels and directions to capture the topology of Q and G; and (d) be monotonically decreasing to ensure the robustness of $\mathsf{VF3}_M$ (see Section 4).

One might want to use existing models for subgraph matching or counting, *e.g.,* GNNPE [118], D2Match [72], SKETCH [129], DMPNN [71], GNCE [93] and SPACE [17]. However, these models do not work here since they (1) do not take partial match ρ of a pattern Q as input, and (2) even when we extend these models to predict ρ, they check only unmatched part of Q without considering what is already matched in ρ (see Section 6 for more details).

Overview. To address these challenges, we design $\mathcal{M}$ as illustrated in Figure 8. Given a graph G, a pattern Q and a partial match ρ (highlighted in red), we (1) compute vertex embeddings of G and Q using IDGNN; (2) partition G and Q into multiple paths; (3) aggregate path embeddings by concatenating their vertex embeddings and assigning distinct weights to vertices depending on whether they belong to the partial match ρ; and (4) perform prediction by comparing path embeddings in the order-embedding space [119].

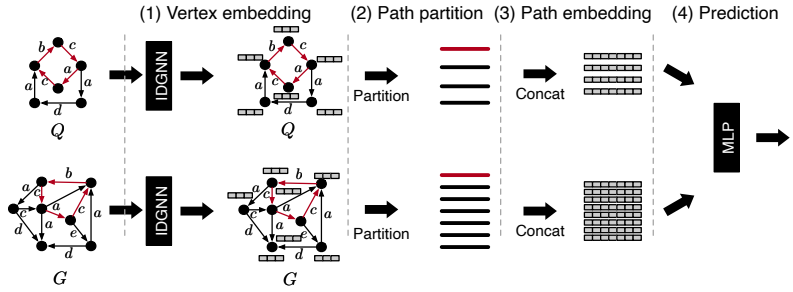


Fig. 8. The framework of $\mathcal{M}$

(1) We employ IDGNN [121] to capture the topology of G and Q. Unlike vanilla GNNs, IDGNN incorporates vertex identities into message passing, improving its ability to distinguish non-isomorphic graphs. We further extend it to handle edge labels and directions.

(2) For efficiency, we pre-compute vertex embeddings of G and Q with IDGNN before enumeration, computing each only once and reusing them across different patterns and partial matches.

(3) We adopt a partition-based strategy from GNNPE [118] to improve the accuracy of $\mathcal{M}$. Intuitively, paths capture topological details of G and Q, such as vertices connected along the same path, thus aiding more accurate predictions. By contrast, existing models such as D2Match, SKETCH, DMPNN, GNCE and SPACE rely on entire-graph embeddings and may miss such structural insights. Different from GNNPE, $\mathcal{M}$ takes partial matches as input.

(4) Given a partial match ρ, we weight vertices according to their inclusion in ρ when computing path embeddings, helping $\mathcal{M}$ preserve the existing mappings. To enforce monotonicity, we adjust the confidence score by a ρ-dependent constant and augment the training set with known invalid partial matches (see below).

5.1 Graph Embedding

We now present steps (1)–(4) one by one.

(1) Vertex embedding. Given a graph G and a vertex v, we exploit GNNs to compute the embedding of v in G, since GNNs can capture structural similarity in the embedding space [115, 118]. More specifically, GNN computes an embedding of v by iteratively aggregating embeddings of its neighbors. At each step, neighbor embeddings are collected via a message-passing function MSG and combined by an aggregation function AGG to update v. These steps repeat m times (*i.e.,* the number of GNN layers). However, such GNNs cannot distinguish certain non-isomorphic graphs and are no more expressive than the 1-dimensional Weisfeiler–Leman (1-WL) test [115].

To improve the accuracy of $\mathcal{M}$, we adopt IDGNN, which extends vanilla GNNs by using distinct MSG functions for different vertices. More expressive than the 1-WL test and vanilla GNN [121], IDGNN can better distinguish non-isomorphic graphs. Specifically, for each vertex v, it (1) collects the m-hop subgraph G_v centered at v, and (2) iteratively computes embeddings of vertices $u \in G_v$, by

$$h_u^0 := x_u, \tag{1}$$

$$h_u^{l+1} := \mathrm{AGG}^l(\{\mathrm{MSG}^l_{\mathbb{I}(u_n=v)}(h_u^l), u_n \in N(u)\}, h_u^l). \tag{2}$$

Here (a) x_u is the initial feature of vertex u in G_v, *e.g.,* the embedding of its label; (b) AGG^l is the aggregation function, which updates the embedding of u by combining its current embedding h_u^l with the information of its neighbors collected through the message passing function $\mathrm{MSG}^l_{\mathbb{I}(u_n=v)}$; (c) for each neighbor u_n of u, $\mathrm{MSG}^l_{\mathbb{I}(u_n=v)}(h_{u_n})$ constructs a message sent to u using the embedding $h^l_{u_n}$ of u_n; and (d) $\mathbb{I}(u_n = v)$ is an indicator function, returning 1 if u_n and v are the same vertex, and 0 otherwise.

Input: Graph G.
Output: A set $\mathcal{S}_G$ of paths containing all vertices in G.
1. pick a vertex v_0 with the maximum degree in G;
2. randomly pick a set $\mathcal{S}_G$ of paths containing v_0;
3. **while** a vertex v in G is not covered by $\mathcal{S}_G$ **do**
4. $\quad \mathcal{S}_G := \mathcal{S}_G \cup \mathsf{Expand}(G, \mathcal{S}_G, v)$;
5. **return** $\mathcal{S}_G$;

Fig. 9. Path partition

Note that the message passing function $\mathsf{MSG}^l_{\mathbb{1}(u_n=v)}$ is not uniform across all vertices in G_v. IDGNN applies message passing functions MSG^l_1 to v and MSG^l_0 to other vertices. To handle edge labels and directions, it assigns distinct weights to edge labels when constructing messages [91], and concatenates embeddings of incoming and outgoing edges when updating u [59].

The subgraph G_v is computed efficiently by neighbor sampling [52] and used only for embedding v [121]. More specifically, (1) we first uniformly sample a fixed number of neighbors of v, denoted N_v, and use the vertices in $N_v \cup \{v\}$ and the edges among them to form G_v; (2) we then repeat this process for $m-1$ iterations to collect the m-hop neighbors of v: in each iteration, let N_n denote the neighbors newly added in the previous step; then we uniformly sample a fixed number of neighbors for each vertex $u \in N_n$, and extend G_v with these sampled vertices.

(2) Path partition. Given a graph G, we partition it into a set of directed paths $\mathcal{S}_G$, ensuring that each vertex appears in at least one path. We adopt the path partition framework, as path embeddings enable exact subgraph matching without missing matches [118]. Since the number of paths in $\mathcal{S}_G$ directly affects the efficiency of the framework (hence more paths lead to slower inference in $\mathcal{M}$), we aim to cover all vertices in G using as few paths as possible.

To reduce cost, we employ a degree-based strategy (Figure 9): (a) select the maximum-degree vertex v_0 (line 1); (b) randomly generate paths of length up to l that include v_0 (line 2), where l balances complexity and accuracy [118]; (c) iteratively add paths to cover uncovered vertices via procedure Expand (lines 3–4), which starts from an uncovered vertex v' and randomly selects N_p edge-directed paths to capture sufficient information, with N_p controlling the same trade-off [52]; and (d) terminate once all vertices are covered (line 5).

Intuitively, starting from the maximum-degree vertex yields longer paths (*i.e.,* paths with more vertices) and thus fewer total paths, improving coverage efficiency [118].

(3) Path embeddings. Given a graph G and its partitioned paths $\mathcal{S}_G$, each path $p \in \mathcal{S}_G$ is embedded by concatenating the embeddings of its vertices. Vertex embeddings are pre-computed with IDGNN (step (1)) and reused across paths. We cannot directly concatenate embeddings of vertices on p as in GNNPE [118], since path p may contain vertices both within and outside the partial match ρ. To preserve the mappings encoded by a partial match ρ, we assign distinct weights to vertices depending on whether they belong to ρ. More specifically, the embedding of a path p is computed as follows:

$$e_p = \sum_{v \in \rho \cap p} c_1 e_v + \sum_{v \in \cap p \setminus \rho} c_2 e_v,$$

where (a) c_1 and c_2 are weights for matched and unmatched vertices, respectively; (b) e_v is the embedding of v; (c) $\rho \cap p$ consists of vertices appearing in both partial match ρ and path p; and (d) $p \setminus \rho$ denotes the set of vertices that appear in p but not in ρ. Intuitively, assigning different weights to vertices inside and outside ρ prevents pattern vertices outside ρ from matching those of G within ρ, and vice versa. The coefficients c_1 and c_2 are learned during the training of $\mathcal{M}$.

Let $\mathcal{S}^e_Q$ (resp. $\mathcal{S}^e_G$) be set of embeddings of paths in $\mathcal{S}_Q$ (resp. $\mathcal{S}_G$), which are extracted from pattern Q (resp. graph G) in step (2).

(4) Prediction. Given $\mathcal{S}_G^e$ and $\mathcal{S}_Q^e$, the task is to determine whether a partial match ρ is valid. We leverage the order-embedding space [119] to compare path embeddings in $\mathcal{S}_G$ and $\mathcal{S}_Q$, which capture the topological structures of G and Q, respectively. For paths $p_q \in \mathcal{S}_Q$ and $p_g \in \mathcal{S}_G$ with embeddings $e_q = (a_1, \ldots, a_d)$ and $e_g = (b_1, \ldots, b_d)$, we require $a_i \leq b_i$ for all i if p_q is a subpath of p_g. Unlike the original formulation [119], which compares only two embeddings, our setting compares two sets of embeddings: each embedding in $\mathcal{S}_Q^e$ is evaluated against all embeddings in $\mathcal{S}_G^e$.

Moreover, when Q is connected, the corresponding selected paths in G must also be connected. Therefore, if every path in $\mathcal{S}_Q$ is a subpath of some path in $\mathcal{S}_G$ and connectivity is preserved, $\mathcal{M}$ outputs true, indicating that ρ can be extended to a full match.

To return the confidence that ρ is valid (Section 3.3), we use a multilayer perceptron (MLP) to output a score in $[0, 1]$. As MLPs take vectors as input, the sets $\mathcal{S}_G^e$ and $\mathcal{S}_Q^e$ are encoded into vectors. A partial match ρ is valid only if each path $p_q \in \mathcal{S}_Q$ is a subpath of some $p_g \in \mathcal{S}_G$ (*i.e.,* $a_i \leq b_i$ for all i). Accordingly, (a) for each p_q, we select a corresponding p_g; (b) rather than requiring $a_i \leq b_i$ strictly, we apply a margin threshold, allowing occasional $a_i > b_i$ to improve recall and robustness of $\mathsf{VF3}_M$ [31, 109, 120]; and (c) we quantify the distance between e_q and e_g using Chebyshev distance $d_c(e_q, e_g) = \max_{i \in [1,d]}(b_i - a_i)$, *i.e.,* the maximum coordinate difference [20].

Putting these together, we compute the embeddings of $\mathcal{S}_G^e$ and $\mathcal{S}_Q^e$ in two steps. (a) For each $e_q \in \mathcal{S}_Q^e$, select $e_g \in \mathcal{S}_G^e$ with the minimum Chebyshev distance, forming a set $\mathcal{S}_{\mathsf{path}}^e = \{\langle e_q, e_g \rangle\}$. (b) Concatenate all e_q (resp. e_g) in $\mathcal{S}_{\mathsf{path}}^e$ to obtain e_Q (resp. e_G).

For example, if $\mathcal{S}_Q^e = \{(1, 2, 3), (4, 3, 1)\}$ and $\mathcal{S}_G^e = \{(3, 1.8, 6), (1, 2, 4), (4, 2.8, 1), (3.9, 3, 1)\}$, we get $\mathcal{S}_{\mathsf{path}}^e = \{\langle (1, 2, 3), (1, 2, 4) \rangle, \langle (4, 3, 1), (3.9, 3, 1) \rangle\}$, yielding the minimum Chebyshev distances of 0 and 0.1. Thus $e_Q = (1, 2, 3, 4, 3, 1)$ and $e_G = (1, 2, 4, 3.9, 3, 1)$.

To calculate the Chebyshev distances, we construct d sorted sets, one for each dimension of the embeddings. For each dimension i, the embeddings in sets $\mathcal{S}_G^e$ and $\mathcal{S}_Q^e$ are sorted by their values at index i. Given an embedding in $\mathcal{S}_Q^e$, we find the embedding in $\mathcal{S}_G^e$ with the minimum Chebyshev distance by inspecting these d sorted sets.

Complexity. We analyze the pre-computation and prediction times.

(1) The pre-computation, executed only once, takes $O(m(|Q| + |G|)(f_e(|G|) + f_e(|Q|)) + l(|G|d_{\max}^G + |Q|d_{\max}^Q))$ time, to (a) compute vertex embeddings for both G and Q, and (b) conduct path partitioning. Here f_e denotes the cost of updating embeddings via AGG and MSG, which is polynomial after $\mathcal{M}$ is trained, and l is the length bound for partitioned paths. Note that (i) the embedding process takes m rounds; (ii) for each round, given a vertex v, $\mathcal{M}$ takes $(|Q| + |G|)(f_e(|G|) + f_e(|Q|))$ time to update v's embedding as in Equation (2); and (iii) path partitioning takes at most $O(l(|G|d_{\max}^G + |Q|d_{\max}^Q))$ time, where $d_{\max}^G$ (resp. $d_{\max}^Q$) is the maximum degree of G (resp. Q).

(2) Given a graph G, a pattern Q and a partial match ρ, the prediction takes $O(d(P_{|Q|}^\rho + P_{|G|}^\rho) + f_m)$ time, where (a) $P_{|Q|}^\rho$ and $P_{|G|}^\rho$ are the numbers of paths in $\mathcal{S}_Q$ and $\mathcal{S}_G$ that include vertices of ρ within Q and G, respectively; (b) d is the dimension of embeddings; and f_m is the time required for prediction via MLP, which is polynomial after $\mathcal{M}$ is trained. Observe that (i) only paths containing vertices in ρ are utilized for prediction, which improves the recall of $\mathcal{M}$, and the numbers $P_{|Q|}^\rho$ and $P_{|G|}^\rho$ of paths are small when l is small. (ii) Computing embeddings of paths takes $O(d(P_{|Q|}^\rho + P_{|G|}^\rho))$ time, due to the use of sorted sets and the locality of pattern matching (see Section 6). (iii) MLP outputs a confidence score in $O(f_m)$ time.

5.2 Training

We next outline the training process for model $\mathcal{M}$. Let D_T denote the training dataset, comprising tuples of the form (G, Q, ρ), where (a) G is a graph and Q is a pattern, and (b) ρ is a partial match of Q in G. Denote by D_T^+ (resp. D_T^-) the set of positive (resp negative) tuples (G, Q, ρ) in D_T such that ρ is valid (resp. invalid). We train model $\mathcal{M}$ by minimizing the following margin loss function [109]:

$$L(D_T) := \sum_{(G,Q,\rho)\in D_T^+} \sum_{p_q \in \mathcal{S}_Q} \min_{p_g \in \mathcal{S}_G} \|\max(0, d_c(e_g, e_q))\|_2^2$$
$$+ \sum_{(G,Q,\rho)\in D_T^-} \sum_{p_q \in \mathcal{S}_Q} \min_{p_g \in \mathcal{S}_G} \max(0, \alpha - \|\max(0, d_c(e_g, e_q))\|_2^2).$$

Here $\|\cdot\|_2$ denotes the L_2-norm, and α is a margin hyperparameter, to control the separation between positive and negative instances, thereby enhancing the generalization of the model [77, 106]. The model $\mathcal{M}$ is trained for N epochs, to improve its accuracy [60].

Data Preparation. We generate the training datasets in three steps. (1) Based on a sampled subgraph G' from the original G, we construct query patterns Q via random walks [101]. Specifically, starting from a randomly selected vertex, we iteratively expand Q by adding neighboring vertices until its size $|Q|$ reaches the threshold τ_Q (set to 20 by default). (2) We then use VF3 to identify partial matches of these patterns in G', marking them as positive if they can be extended to a full match, and negative otherwise. (3) Finally, we sample an equal number of positive and negative samples.

Optimization. Training $\mathcal{M}$ is modest, as it only learns the embedding model IDGNN and the prediction model. As will be seen in Section 6, $\mathcal{M}$ is fairly accurate with only 5000 epochs in 1.6h.

We accelerate training by using fewer epochs; *e.g.,* training can stop early when the loss does not decrease for 10 consecutive epochs [127], which improves model generalization [6]. Additional acceleration strategies include: (1) GNN simplification, which decouples embedding combination from the update phase in function AGG to avoid message passing [41, 74]; (2) vertex-cluster sampling, which samples m-hop neighborhoods based on vertex clusters to enhance sampling efficiency [125]; (3) data- and hardware-aware execution planning, which identifies the optimal execution plan to minimize makespan [69]; and (4) multi-GPU parallelism [103].

Moreover, training $\mathcal{M}$ can be accelerated by using a set $\mathcal{Q}'$ of representative patterns. We construct $\mathcal{Q}'$ as follows: (1) initialize $\mathcal{Q}'$ with K patterns randomly selected from all sampled patterns, where K is a parameter; (2) cluster patterns in $\mathcal{Q}$ using graph similarity measures [94], *i.e.,* each pattern is assigned to the cluster having the minimum similarity measures; (3) in each cluster, select the pattern having the minimum similarity measure with other patterns to update $\mathcal{Q}'$; repeat steps (2) and (3) until $\mathcal{Q}'$ becomes stable.

Implementation. The model $\mathcal{M}$ is trained in PyTorch and evaluated on CPU [66]. After training, we extract all model weights and perform inference directly on the CPU: given a graph G and a pattern Q, vertex embeddings, path embeddings, and predictions are computed via matrix multiplications using the extracted weights.

We also support evaluation in C++ with SIMD (Single Instruction Multiple Data, *e.g.,* AVX/AVX2 [57]) or GPU (*e.g.,* CUDA [32]). However, GPU implementation does not improve inference speed, as data transfer between CPU and GPU memory is slow [96, 104] and incurs additional overhead (see Section 6). To further reduce prediction cost, we can implement the VF3_M algorithms in C++ with AVX/AVX2 instructions rather than CUDA (see Exp-3).

Remark. To make $\mathcal{M}$ monotonically decreasing, we adopt two strategies. (a) We adjust the confidence score by adding $e^{-|\rho|}/2$, where $|\rho|$ is the number of vertices in partial match ρ. Intuitively,

larger $|\rho|$ yields a smaller adjustment, leading to lower confidence. (b) We enrich training set D_T with additional negatives. For each tuple $(G, Q, \rho) \in D_T^-$, we add $K{=}C_s(|Q|{-}|\rho|)$ new invalid tuples $(G, Q, \rho_1), \ldots, (G, Q, \rho_K)$, where each ρ_i extends ρ by selecting C_s candidates from $C(u)$ for unmatched vertices u [78], for a parameter C_s. Intuitively, we pick C_s vertices from candidate set $C(u)$ for each vertex u that has no match in ρ, to extend the invalid match ρ.

6 Experimental Study

Using real-life and synthetic datasets, we experimentally evaluated $\mathsf{VF3}_M$ algorithms for their (1) accuracy and (2) efficiency and scalability. We also evaluated (3) the performance of model $\mathcal{M}$.

Experimental setting. We start with the experimental setting.

Datasets. We used four real-life graphs: (1) DBLP [1], a citation network with 0.3M vertices and 1M edges; and (2) Pokec [3], an on-line social network in Slovakia with 1.6M vertices and 30M edges; (3) Patents [67], a U.S. patent citation graph (1975–1999) with 3.7M nodes and 15M edges; and (4) LiveJournal [2], a free on-line community graph with over 3M vertices and 60M edges.

To evaluate the scalability of algorithms, we generated synthetic graphs $G = (V, E, L)$ using datagen [58], controlled by the number $|V|$ of vertices (up to 4M) and the number $|E|$ of edges (up to 70M).

Patterns. For each graph, we generated 100 patterns following a sampling method [16, 128]. For each graph G, we first conducted random walks in G to identify vertices, and then extracted the subgraphs induced by these vertices. We treated these subgraphs as patterns.

Algorithms. We implemented in C++ (1) $\mathsf{VF3}_M^N$, $\mathsf{VF3}_M^O$ and $\mathsf{VF3}_M^D$ (Section 3.3); and (2) a variant $\mathsf{VF3}_O$ of VF3 with optimizations in Section 3.1 but without using $\mathcal{M}$. We compared with six algorithms as baselines: (3) VF3 [4], the classic VF3 algorithm; (4) CECI [21], (5) RM [100], (6) DPiso [53]; these three work just like VF3, but use different matching orders, filtering strategies and index structures; (7) GNNPE [118], which applies pruning strategies guided by GNNs; and (8) MLSearch [116], which enhances VF3 by leveraging the embeddings of Q and G to predict whether a valid match exists.

We compared our model $\mathcal{M}$ (Section 5) with the following: (9) D2Match [72], which employs deep GNNs and degeneracy for subgraph matching; (10) MS [116], which is the prediction model in algorithm MLSearch, and adopts graph attention network for subgraph matching; and (11) DMPNN [71], which develops dual message passing neural network for subgraph counting and matching. None of these models supports partial matches ρ. For example, given a pattern Q and a graph G, D2Match directly assesses the existence of a full match of Q in G. For a fair comparison, we extend these models as follows: given ρ, first generate pattern Q^r and graph G^r from Q and G by removing all vertices in ρ, respectively, and then apply these models on Q^r and G^r.

Environment. We ran experiments on a machine with 2 Intel Xeon Gold 6148 CPUs @ 2.40GHz, 504 GB memory, and 8 Tesla V100 GPUs (32 GB each). For fairness, both SubIso algorithms and model predictions were executed on CPU, while only ML models were trained on GPU. By default, we set $|Q| = |V_Q| + |E_Q| = 20$, $\delta_{\mathsf{T}} = 0.7$, and $\delta_{\mathsf{F}} = 0.5$ (confidence thresholds for $\mathcal{M}$ in $\mathsf{VF3}_M$, Section 3.3). Each test was run 5 times and averaged; we report only representative results, with other datasets exhibiting consistent trends.

Experimental findings. We next report our findings.

Exp-1: Accuracy. We first report the F1 measure, consistency and robustness of $\mathsf{VF3}_M^N$, $\mathsf{VF3}_M^O$ and $\mathsf{VF3}_M^D$. Since CECI, RM, DPiso, GNNPE and VF3 return all full matches, their F1 scores are 1.

F1 score. As shown in Table 2, on average, (1) the F1 score of $\mathsf{VF3}_M^N$, $\mathsf{VF3}_M^O$ and $\mathsf{VF3}_M^D$ is 0.44, 0.95

Method	DBLP		Patents		Pokec		LiveJournal	
	F1	Time (s)	F1	Time (s)	F1	Time (s)	F1	Time (s)
RM	1	7.72 (48.25×)	1	3.03 (2.50×)	1	33.28 (17.06×)	1	77.79 (27.59×)
CECI	1	3.88 (24.25×)	1	1.69 (1.39×)	1	14.62 (7.50×)	1	34.45 (12.21×)
DPiso	1	1.93 (12.06×)	1	1.36 (1.12×)	1	12.74 (6.53×)	1	21.60 (7.66×)
VF3	1	2.52 (15.75×)	1	4.14 (3.42×)	1	13.50 (6.92×)	1	21.21 (7.52×)
$VF3_O$	1	0.17 (1.06×)	1	2.01 (1.66×)	1	2.56 (1.31×)	1	3.98 (1.41×)
GNNPE	1	0.22 (1.38×)	1	2.59 (2.14×)	1	2.24 (1.15×)	1	DNP
MLSearch	0.86	1066.55 (6665×)	*	Timeout	*	Timeout	*	Timeout
$VF3^D_M$	0.99	0.16	0.98	1.21	0.98	1.95	0.97	2.82
$VF3^N_M$	0.54	0.12 (↑ 25%)	0.23	1.13 (↑ 6.6%)	0.62	1.90 (↑ 2.5%)	0.38	2.20 (↑ 21.9%)
$VF3^O_M$	0.99	0.13 (↑ 18.8%)	0.94	1.18 (↑ 2.4%)	0.93	1.95 (↑ 0%)	0.94	2.43 (↑ 13.8%)

Table 2. Accuracy and runtime; '*' indicates unavailable due to timeout (>6h), and DNP (Did Not Preprocess) denotes failure due to excessive preprocessing time (>6h)

and 0.98, up to 0.62, 0.99 and 0.99, respectively. (2) $VF3^D_M$ is 3% (resp. 54%) more accurate than $VF3^O_M$ (resp. $VF3^N_M$), since it employs DeepCheck to reduce FNs.

Consistency. We tested $F1(VF3_M)$ when the embedded ML model $\mathcal{M}$ is error free, *i.e.,* when $\eta = 0$ (Section 4). To this end, we stored in $Q(G)$ all full matches found by VF3, and determined whether a partial match ρ is valid by checking whether there exists a full match in $Q(G)$ extended from ρ, *i.e.,* by treating $Q(G)$ as an ideal model $\mathcal{M}$. We find that (a) all returned matches are valid, *i.e.,* the precision of these algorithms is 1, and (b) all matches in $Q(G)$ are returned by the three, *i.e.,* the recall of these algorithms is also 1. Thus $VF3^N_M$, $VF3^O_M$ and $VF3^D_M$ are consistent. These empirically confirm Theorem 3.

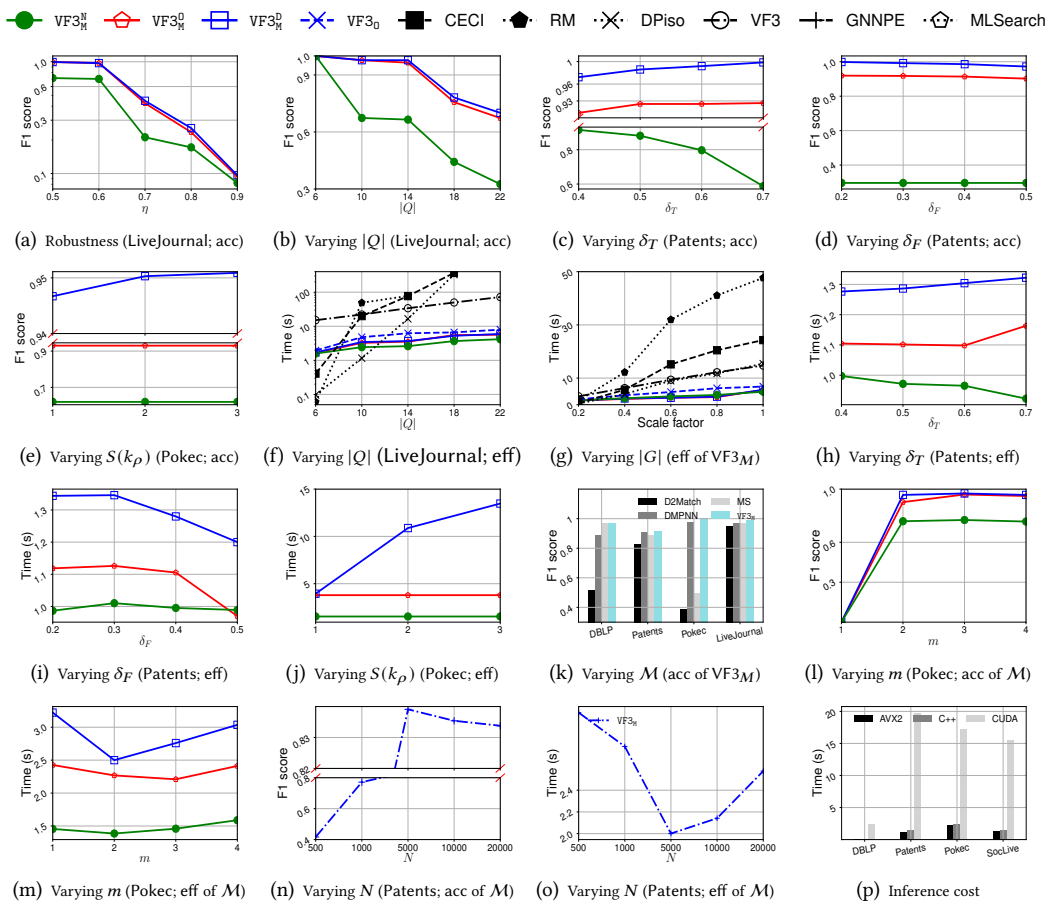
Robustness. We next tested the robustness. We used five models $\mathcal{M}$ with 0.5, 0.6, 0.7, 0.8 and 0.9 as error rate η. As shown in Figure 10(a): (1) with smaller η (*i.e.,* more accurate $\mathcal{M}$), all $VF3_M$ variants get more accurate and faster due to reduced redundancy (Theorem 6). (2) At $\eta = 0.5$, F1 scores of $VF3^D_M$, $VF3^O_M$ and $VF3^N_M$ are 0.99, 0.99 and 0.71, respectively; when $\eta = 0.6$, *i.e.,* when $\mathcal{M}$ performs worse than a coin toss, $VF3^D_M$ and $VF3^O_M$ still have F1 scores above 0.9. These are consistent with Theorems 4 and 5. (3) $VF3^D_M$ is the most robust; even at $\eta = 0.7$, its F1 score is 0.45, outperforming $VF3^O_M$ (0.43) and $VF3^N_M$ (0.21), as DeepCheck improves recall.

The impact of parameters. We also evaluated the impact of $|Q|$, δ_T and δ_F on the F1 score of $VF3^N_M$, $VF3^O_M$ and $VF3^D_M$.

(1) Varying $|Q|$. We varied the pattern size $|Q|$ from 6 to 22. As shown in Figure 10(b), (1) as $|Q|$ increases, the F1 scores of $VF3^N_M$, $VF3^O_M$ and $VF3^D_M$ generally decrease, since larger patterns make valid partial matches harder to identify and reduce ML prediction confidence. (2) $VF3^D_M$ outperforms $VF3^O_M$ and $VF3^N_M$ by 5% and 76% in accuracy, respectively, as its use of DeepCheck reduces FNs.

(2) Varying δ_T. We evaluated the impact of confidence threshold δ_T. As shown in Figure 10(c), (1) when δ_T varies from 0.4 to 0.7, the F1 score of $VF3^N_M$ decreases, since not all valid partial matches get high confidence when $\mathcal{M}$ is not error-free, and when δ_T gets larger, more valid matches may be missed. (2) $VF3^O_M$ and $VF3^D_M$ are less sensitive to δ_T since they invoke IsFeasible to reduce FPs, no matter whether their ML models are δ_T-confident or not.

(3) Varying δ_F. We varied the confidence threshold δ_F for false, as shown in Figure 10(d): (1) as δ_F increases from 0.2 to 0.5, the F1 scores of $VF3^O_M$ and $VF3^D_M$ decrease, since $\mathcal{M}$ returns false on more partial matches. (2) $VF3^D_M$ is more sensitive to δ_F than to δ_T, as larger δ_F may exclude more partial matches. (3) $VF3^N_M$ remains unaffected by δ_F since it does not rely on this parameter (Section 3.3).



(a) Robustness (LiveJournal; acc) (b) Varying $|Q|$ (LiveJournal; acc) (c) Varying δ_T (Patents; acc) (d) Varying δ_F (Patents; acc)

(e) Varying $S(k_\rho)$ (Pokec; acc) (f) Varying $|Q|$ (LiveJournal; eff) (g) Varying $|G|$ (eff of VF3$_M$) (h) Varying δ_T (Patents; eff)

(i) Varying δ_F (Patents; eff) (j) Varying $S(k_\rho)$ (Pokec; eff) (k) Varying $\mathcal{M}$ (acc of VF3$_M$) (l) Varying m (Pokec; acc of $\mathcal{M}$)

(m) Varying m (Pokec; eff of $\mathcal{M}$) (n) Varying N (Patents; acc of $\mathcal{M}$) (o) Varying N (Patents; eff of $\mathcal{M}$) (p) Inference cost

Fig. 10. Performance evaluation (acc for accuracy and eff for efficiency)

Varying $S(k_\rho)$. We varied the number of deep checking from $S(k_\rho)$ to $3S(k_\rho)$. As shown in Figure 10(e), (1) when $S(k_\rho)$ gets larger, $F1(\text{VF3}^D_M)$ increases, as expected. (2) $F1(\text{VF3}^D_M)$ becomes stable once the number reaches $2S(k_\rho)$, since ρ is invalid with high probability when no match is identified after $2S(k_\rho)$ many deep checking.

Exp-2: Efficiency and scalability. We next show that the VF3$_M$ algorithms indeed speed up the enumeration process for SubIso. The results are consistent with Theorem 6, Corollary 7 and Theorem 8 for output-polynomial properties and competitive ratios.

Efficiency. As shown in Table 2, (1) VF3^D_M (in yellow) beats RM, CECI, DPiso, GNNPE, VF3 and VF3$_O$ by 22.49×, 10.74×, 6.47×, 1.56×, 8.40× and 1.36×, respectively. MLSearch is 6,665× slower than VF3^D_M on DBLP and fails to terminate on other graphs after 6 hours. (2) VF3^D_M is only 14% and 8.8% slower than VF3^O_M and VF3^N_M (marked as ↑), respectively, but is more accurate due to its deep checking. VF3 beats RM and CECI since (a) RM is join-based algorithm, which produces large intermediate results on dense graphs, and (b) CECI has a threshold for early termination, which we removed to return all matches, degrading the performance of CECI.

Varying $|Q|$. We varied $|Q|$ from 6 to 22. As shown in Figure 10(f), (1) VF3^D_M is faster than all the baselines; *e.g.*, when $|Q| = 14$, it is 21.15×, 20.86×, 4.49×, 9.21× and 1.69× faster than RM, CECI,

DPiso, VF3 and VF3$_O$, respectively; hence, ML models in VF3$_M$ effectively reduce false positives and runtime. When $|Q| = 22$, VF3$_M^D$ (resp. VF3$_M^O$) is 1.34× (resp. 1.40×) faster than VF3$_O$, the fastest baseline. (2) VF3$_M^D$ is almost as efficient as VF3$_M^O$, despite additional DeepCheck calls. Algorithm DPiso's runtime rises quickly when increasing $|Q|$, due to the overhead of building its failing sets to reduce the search space. GNNPE fails on LiveJournal with more than 60M edges.

Varying $|G|$. Fixing $|Q| = 22$, $\delta_T = 0.7$, and $\delta_F = 0.5$, we varied the scale of synthetic graphs $G = (V, E)$ from 0.2 to 1.0. As shown in Figure 10(g): (1) all three VF3$_M$ algorithms naturally slow down as $|G|$ increases; (2) VF3$_M^N$, VF3$_M^O$ and VF3$_M^D$ scale well: on graphs with 4M vertices and 70M edges, they complete in 4.89s, 5.58s and 5.59s, respectively, achieving F1 scores of 0.36, 0.94, and 0.97; and (3) VF3$_M^D$ is up to 13.61×, 6.78×, 3.90×, 4.04× and 2.05× faster than RM, CECI, DPiso, VF3 and VF3$_O$, respectively. Results for GNNPE and MLSearch are omitted due to excessive preprocessing times (>6h).

Varying δ_T. As shown in Figure 10(h), (1) on Patents, when δ_T increases from 0.4 to 0.7, the competitive ratio of VF3$_M^N$ gets smaller. This is because VF3$_M^N$ only returns valid partial matches with high confidence, and when δ_T increases, the number of such partial matches decreases. (2) In contrast, VF3$_M^O$ and VF3$_M^D$ are insensitive to δ_T, since they use IsFeasible as a backup regardless of δ_T.

Varying δ_F. As shown in Figure 10(i), (1) VF3$_M^N$ is indifferent to δ_F as it does not use this parameter. (2) When δ_F increases from 0.2 to 0.5, both VF3$_M^O$ and VF3$_M^D$ get faster, as they inspect fewer candidate matches. (3) In this setting, VF3$_M^D$ is on average 16% slower than VF3$_M^O$, as with larger δ_F, VF3$_M^D$ invokes DeepCheck more often.

Varying $S(k_\rho)$. We varied the number of deep checks from $S(k_\rho)$ to $3S(k_\rho)$. As shown in Figure 10(j), (1) as the number increases, VF3$_M^D$ gets slower, as expected. (2) VF3$_M^D$ takes 0.2s longer when the number varies from $S(k_\rho)$ to $2S(k_\rho)$, but its accuracy improves (see Fig. 10(e)), which justifies the design of VF3$_M^D$ (see Section 3.3).

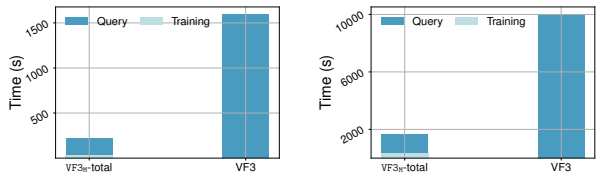
Exp-3: ML model. We evaluated the impact of model $\mathcal{M}$ (Section 5) on the accuracy of VF3$_M$ and the cost to train $\mathcal{M}$. Unless stated otherwise, we fixed $m = 2$ and $N = 5000$.

Impact on VF3$_M$ algorithms. We tested VF3$_M$ variants that take D2Match, MS and DMPNN as ML model $\mathcal{M}$, respectively. As shown in Figure 10(k), on average VF3$_M^N$ outperforms these variants by 0.61, 0.05 and 0.07 in F1-score, respectively. This verifies that model $\mathcal{M}$ of Section 5 is more accurate than D2Match, MS and DMPNN.

Varying m. We varied the number of IDGNN layers m from 1 to 4. As shown in Figure 10(l) and 10(m) on Pokec, the accuracy of VF3$_M$ first improves and then remains stable; optimal performance is observed at m=2, since (a) small m provides insufficient context; and (b) large m causes over-smoothing, making distant neighbors indistinguishable [115]. VF3$_M$ also becomes the fastest when $m = 2$.

Varying N. We tested the impact of the number (N) of training epochs. As shown in Figures 10(n) and 10(o), VF3$_M$ achieves the highest accuracy when $N = 5000$. This is because (1) when N is too small, $\mathcal{M}$ is under-trained with near-random parameters; but (2) when N is too large, overfitting occurs, reducing generalization on unseen patterns. VF3$_M$ becomes the fastest when $N = 5000$ by using a more accurate model to filter invalid candidates.

Training cost. (1) Training is modest: with m=2 and N=5000, $\mathcal{M}$ converges in 1.6h (not shown). (2) The cost can be reduced with fewer epochs or smaller data; *e.g.,* on Patents, (a) VF3$_M$ still achieves 0.82 F1 with 2000 epochs, converging in 10 minutes; (b) representative patterns reduce training time by 66% and achieve a F1-score comparable to models trained with sampled patterns (0.92 vs. 0.95).



(a) Amortized analysis(DBLP) (b) Amortized analysis (Pokec)

Fig. 11. Amortized analysis

Prediction cost. We compared the impact of different implementations of $\mathcal{M}$ on VF3$_M$. We implemented ML prediction using (1) AVX/AVX2 instructions, (2) standard C++ and (3) CUDA, after extracting model weights from $\mathcal{M}$. As shown in Figure 10(p), VF3$_M$ implemented with AVX/AVX2 and standard C++ outperform the one with CUDA. This is because the model is lightweight and runs efficiently on CPU, while CUDA transfers data between CPU and GPU.

Amortized analysis. We performed an amortized cost analysis of VF3$_M$, where the model $\mathcal{M}$ is trained once offline and then reused to answer 1000 queries without retraining. This setting is practical; for example, marketing platforms such as Zeta Global handle over 450 million queries daily [113]. We compared VF3$_M$-total, which includes both training and query time, with the original VF3 algorithm. As shown in Figure 11: (1) training $\mathcal{M}$ accounts for only a small fraction of the total time of VF3$_M$-total, *e.g.,* on DBLP and Pokec, just 12% and 19%, respectively; and (2) even including training, VF3$_M$-total remains on average 6.67× faster than VF3.

Summary. We find the following. *(1) Speedup.* Using ML oracles, VF3$_M^D$ (resp. VF3$_M^N$, VF3$_M^O$) is 22.49×, 10.74×, 6.47×, 1.56× and 8.40× (resp. 29.97×, 14.29×, 8.45×, 1.73×, 10.35×; and 27.76×, 13.23×, 7.85×, 1.68×, 9.64×) faster than algorithms RM, CECI, DPiso, GNNPE and VF3, respectively, up to 48.25×, 24.25×, 12.06×, 2.14× and 15.75× (resp. 64.33×, 32.33×, 16.08×, 2.29×, 21.0×; and 59.38×, 29.84×, 14.84×, 2.19×, 19.38×). *(2) Accuracy.* The average F1 scores of VF3$_M^N$, VF3$_M^O$ and VF3$_M^D$ are 0.44, 0.96 and 0.98, respectively, all with precision 1. Even when the error rate of $\mathcal{M}$ reaches 0.5, the F1 scores of VF3$_M^O$ and VF3$_M^D$ remain above 0.9. *(3) The accuracy of the ML model.* Our ML model $\mathcal{M}$ outperforms D2Match, MLSearch and DMPNN by 0.61, 0.05 and 0.07 in F1 score, respectively. *(4) Parameter setting.* Optimal performance is observed at $\delta_\mathsf{T} = 0.75$ and $\delta_\mathsf{F} = 0.35$, striking a balance between the accuracy and efficiency. To train model $\mathcal{M}$, N=5000 and m=2 work the best.

7 Related Work

We characterize the related work as follows.

Learning algorithms. Recent studies have explored using ML models for solving optimization problems. (1) Most approaches employ deep reinforcement learning to incrementally compute solutions by optimizing objective functions [63, 87], enhancing ant colony optimization [117], or leveraging collaborative policies [65], encoder–decoder architectures [75], heuristic rules [123] and force-directed methods [79]. (2) Non-reinforcement methods reformulate optimization as sampling [99, 102, 124]: Sun et al. [99] sample high-quality solutions via Markov Chain Monte Carlo; DIFUSCO [102] employs graph-based denoising diffusion models to generate candidate solutions; and Zhang et al. [124] adopt Markov decision processes, using generative flow networks to sample solutions.

Closer to this work are SKETCH [129], DMPNN [71], GNCE [93], D2Match [72] and SPACE [17]. SKETCH [129] adopts active learning with GNNs for subgraph counting. DMPNN [71] employs dual message-passing neural networks for subgraph isomorphism counting. D2Match [72] combines deep GNNs with graph degeneracy, transforming subgraph matching into subtree matching. GNCE [93] integrates knowledge graph embeddings and GNNs to estimate the cardinality of conjunctive queries, while SPACE [17] models path-pattern cardinality by encoding node–edge label sequences.

This work differs from prior studies as follows. (1) We unify algorithmic methods and ML models, leveraging both rather than relying solely on ML [17, 63, 87, 93]. (2) We focus on enumeration algorithms for an EnumP-complete problem, unlike prior work on decision or optimization tasks [79, 102]. (3) We design task-specific ML models for SubIso that scale to large graphs and patterns, rather than general models limited to small graphs [63, 71, 72, 129]. (4) Unlike SKETCH [129], DMPNN [71] and D2Match [72], our model explicitly verifies partial match validity. (5) We further establish evaluation criteria and theoretical guarantees ensuring robustness and accuracy under ML errors, an aspect unaddressed in prior work.

Algorithms with ML oracles. There is also work on algorithms augmented with ML predictions to improve efficiency, grouped as follows. (1) Online algorithms, including online Steiner tree [114], bin packing [13], TSP [48], assignment [108], contract scheduling [12], paging and caching [14, 61, 73], frequency estimation [5], index structures [66], revenue-maximizing auctions [33, 81], and sorting [18]. (2) Design principles for prediction-augmented algorithms: Kumar et al. [86] formalize ML-assisted design with guarantees (*e.g.,* ski rental), Khalil et al. [63] learn greedy heuristics via RL and graph embeddings, and Anand et al. [10] propose regression-based strategies with provable competitive bounds. (3) Performance enhancement strategies, such as incorporating optimization-aware loss functions [11], combining multiple predictors [15, 38, 51, 76], and minimizing prediction cost under known input distributions [36, 40].

Closer to this work are RL-QVQ [110], MLSearch [116], GNNPE [118] and OptMatch [112]. RL-QVQ employs reinforcement learning to generate high-quality matching orders for subgraph enumeration. MLSearch trains a GNN to rank partial matches by degree and label, using sampling to prevent training collapse. GNNPE studies enumeration for SubIso, by partitioning graph patterns into paths and pruning candidates via GNN-based embeddings. OptMatch leverages partial embeddings to efficiently locate initial matches.

This work departs from prior studies in the following. (1) We address enumeration algorithms, rather than finding a single solution for decision or optimization tasks. (2) We focus on an EnumP-complete problem, unlike prior work on tractable (*e.g.,* sorting [18]) or NP-complete tasks (*e.g.,* TSP [48]). (3) We analyze consistency and robustness to characterize the accuracy of $\mathsf{VF3}_M$, beyond measuring runtime or cost as in RL-QVQ [110], GNNPE [118], MLSearch [116] and OptMatch [112]. (4) We extend partition-based strategies to improve $\mathcal{M}$'s accuracy in predicting partial matches, unlike MLSearch [116], which relies on global embeddings, and OptMatch [112], which aggregates node features via attention. (5) We enhance backtrack-based subgraph enumeration with ML oracles, while MLSearch [118] focuses on join-based filtering via GNNs.

Algorithms for SubIso. Prior work on SubIso algorithms falls into three main categories. (1) Exact algorithms, *e.g.,* CECI [21], DPiso [53], VEQ [64], RM [100], CFL [22] and PathLAD+ [111], operate similarly to VF3 (Section 3.1), differing mainly in (a) matching order selection, (b) candidate generation, and (c) index structures (see the recent survey [128]). Unlike VF3, CSCE [26] and GNNPE [118] perform enumeration via join-based frameworks after candidate filtering. (2) Approximate matching methods first select vertex candidates using similarity metrics such as score functions [130], Jaccard or chi-square statistics [42], or significance measures [7], and then compute approximate matches by edge/vertex removal [88], neural distance alignment [90], spanning-tree composition [126], match covers [95], or genetic algorithms [19]. (3) Enumeration speedup techniques improve scalability through parallel or distributed computation, *e.g.,* PBE [49], NemoGPU [70], RPS [50], FAST [55], and PARSEC [39]; $\mathsf{HybridEnum}^+$ [54] and DistriEnum [122] for path patterns; DegCol, DegenCol and DDegCol [68] for clique listing; and IVE [62] for matching isolated vertices via bipartite matching.

Our algorithms for SubIso differ from prior methods in three aspects. (1) We design enumeration

algorithms enhanced with ML predictors, instead of exhaustively enumerating all candidate matches as in exact methods such as CECI [21] and DPiso [53]. (2) The algorithms always return exact matches regardless of predictor errors, unlike approximate approaches [8, 130]. (3) We provide theoretical guarantees (including consistency and robustness) for ML assisted enumeration, a dimension unexplored in prior work.

8 Conclusion

This work advocates an approach to enumeration problems by unifying algorithmic methods and ML predictions. (1) We show that SubIso is one of the hardest problems in EnumP, and for such problems, it is beyond reach in practice to find an algorithm that is both output-polynomial and β-robust for a constant β unless fewP = P. (2) This said, we develop $\mathsf{VF3}_M$ algorithms that extend VF3 with an ML model $\mathcal{M}$ to predict partial matches. We show that these algorithms are consistent, robust with a high probability when $\mathcal{M}$ makes bad predictions, and output-polynomial with a high probability. (3) We train such a model $\mathcal{M}$ via order-embedding space. (4) Our empirical study has verified that the $\mathsf{VF3}_M$ algorithms work well, and the approach is promising since ML techniques evolve.

One topic for future work is to further improve the accuracy of model $\mathcal{M}$ by fine-tuning for different types of graphs and queries. Another topic is to extend the study to other problems in EnumP.

Acknowledgments. The work is supported by National Natural Science Foundation of China (No. U24B20143 and No. 62372030), and State Key Laboratory of Complex & Critical Software Environment (SKLCCSE).

References

[1] 2021. DBLP collaboration network. *https://snap.stanford.edu/data/com-DBLP.html.*
[2] 2025. LiveJournal. *https://snap.stanford.edu/data/soc-LiveJournal1.html.*
[3] 2025. Pokec. *http://snap.stanford.edu/data/soc-pokec.html.*
[4] 2025. vf3lib. *https://github.com/MiviaLab/vf3lib.*
[5] Anders Aamand, Justin Y. Chen, Huy Lê Nguyen, Sandeep Silwal, and Ali Vakilian. 2023. Improved Frequency Estimation Algorithms with and without Predictions. In *NeurIPS*.
[6] Soumyadipta Acharya, Francesco Tenore, Vikram Aggarwal, Ralph Etienne-Cummings, Marc H Schieber, and Nitish V Thakor. 2008. Decoding individuated finger movements using volume-constrained neuronal ensembles in the M1 hand area. *IEEE Transactions on Neural Systems and Rehabilitation Engineering* 16, 1 (2008), 15–23.
[7] Shubhangi Agarwal, Sourav Dutta, and Arnab Bhattacharya. 2021. VerSaChI: Finding Statistically Significant Subgraph Matches using Chebyshev's Inequality. In *CIKM*. 2812–2816.
[8] Shubhangi Agarwal, Sourav Dutta, and Arnab Bhattacharya. 2024. VeNoM: Approximate Subgraph Matching with Enhanced Neighbourhood Structural Information. In *COMAD/CODS*. 18–26.
[9] Eric Allender and Roy S. Rubinstein. 1988. P-Printable Sets. *SIAM J. Comput.* 17, 6 (1988), 1193–1202.
[10] Keerti Anand, Rong Ge, Amit Kumar, and Debmalya Panigrahi. 2021. A Regression Approach to Learning-Augmented Online Algorithms. In *NeurIPS*. 30504–30517.
[11] Keerti Anand, Rong Ge, and Debmalya Panigrahi. 2020. Customizing ML Predictions for Online Algorithms. In *ICML*, Vol. 119. 303–313.
[12] Spyros Angelopoulos and Shahin Kamali. 2023. Contract Scheduling with Predictions. *J. Artif. Intell. Res.* 77 (2023), 395–426.
[13] Spyros Angelopoulos, Shahin Kamali, and Kimia Shadkami. 2023. Online Bin Packing with Predictions. *J. Artif. Intell. Res.* 78 (2023), 1111–1141.
[14] Antonios Antoniadis, Joan Boyar, Marek Eliás, Lene Monrad Favrholdt, Ruben Hoeksma, Kim S. Larsen, Adam Polak, and Bertrand Simon. 2023. Paging with Succinct Predictions. In *ICML*, Vol. 202. 952–968.
[15] Antonios Antoniadis, Christian Coester, Marek Eliás, Adam Polak, and Bertrand Simon. 2023. Mixing Predictions for Online Metric Algorithms. In *ICML*, Vol. 202. 969–983.
[16] Junya Arai, Yasuhiro Fujiwara, and Makoto Onizuka. 2023. GuP: Fast Subgraph Matching by Guard-based Pruning. In *SIGMOD*.

[17] Mehmet Aytimur, Theodoros Chondrogiannis, and Michael Grossniklaus. 2025. SPACE: Cardinality Estimation for Path Queries Using Cardinality-Aware Sequence-based Learning. *Proc. ACM Manag. Data* 3, 3 (2025), 218:1–218:26.
[18] Xingjian Bai and Christian Coester. 2023. Sorting with Predictions. In *NeurIPS*.
[19] Thomas Bärecke and Marcin Detyniecki. 2007. Combining exhaustive and approximate methods for improved sub-graph matching. In *Progress in Pattern Recognition*. 17–26.
[20] Aurélien Bellet, Amaury Habrard, and Marc Sebban. 2015. *Metric learning*. Morgan & Claypool Publishers.
[21] Bibek Bhattarai, Hang Liu, and H. Howie Huang. 2019. CECI: Compact Embedding Cluster Index for Scalable Subgraph Matching. In *SIGMOD*. 1447–1462.
[22] Fei Bi, Lijun Chang, Xuemin Lin, Lu Qin, and Wenjie Zhang. 2016. Efficient Subgraph Matching by Postponing Cartesian Products. In *SIGMOD*. 1199–1214.
[23] Vincenzo Bonnici, Rosalba Giugno, Alfredo Pulvirenti, Dennis E. Shasha, and Alfredo Ferro. 2013. A subgraph isomorphism algorithm and its application to biochemical data. *BMC Bioinform.* 14, S-7 (2013), S13.
[24] Joan Boyar, Lene M. Favrholdt, Christian Kudahl, Kim S. Larsen, and Jesper W. Mikkelsen. 2017. Online Algorithms with Advice: A Survey. *ACM Comput. Surv.* 50, 2 (2017), 19:1–19:34.
[25] Joan Boyar, Lene M. Favrholdt, and Kim S. Larsen. 2018. Relative Worst-Order Analysis: A Survey. In *Adventures Between Lower Bounds and Higher Altitudes - Essays Dedicated to Juraj Hromkovič on the Occasion of His 60th Birthday*. 216–230.
[26] Hongtai Cao, Qihao Wang, Xiaodong Li, Matin Najafi, Kevin Chen-Chuan Chang, and Reynold Cheng. 2024. Large Subgraph Matching: A Comprehensive and Efficient Approach for Heterogeneous Graphs. In *ICDE*. 2972–2985.
[27] Florent Capelli and Yann Strozecki. 2017. On The Complexity of Enumeration. *CoRR* abs/1703.01928 (2017). http://arxiv.org/abs/1703.01928
[28] Vincenzo Carletti, Pasquale Foggia, Alessia Saggese, and Mario Vento. 2017. Introducing VF3: A New Algorithm for Subgraph Isomorphism. In *International Workshop on Graph-Based Representations in Pattern Recognition*. 128–139.
[29] Vincenzo Carletti, Pasquale Foggia, Alessia Saggese, and Mario Vento. 2018. Challenging the Time Complexity of Exact Subgraph Isomorphism for Huge and Dense Graphs with VF3. *IEEE Trans. Pattern Anal. Mach. Intell.* 40, 4, 804–818.
[30] George Casella and Roger Berger. 2001. *Statistical Inference*. Duxbury Resource Center.
[31] Bahzad Charbuty and Adnan Abdulazeez. 2021. Classification based on decision tree algorithm for machine learning. *Journal of Applied Science and Technology Trends* 2, 01 (2021), 20–28.
[32] John Cheng, Max Grossman, and Ty McKercher. 2014. *Professional CUDA c programming*. John Wiley & Sons.
[33] Richard Cole and Tim Roughgarden. 2015. The Sample Complexity of Revenue Maximization. *CoRR* abs/1502.00963 (2015). arXiv:1502.00963 http://arxiv.org/abs/1502.00963
[34] Stephen A. Cook. 1971. The Complexity of Theorem-Proving Procedures. In *STOC*. 151–158.
[35] Nikhil R. Devanur and Thomas P. Hayes. 2009. The adwords problem: Online keyword matching with budgeted bidders under random permutations. In *ACM Conference on Electronic Commerce (EC)*. 71–78.
[36] Ilias Diakonikolas, Vasilis Kontonis, Christos Tzamos, Ali Vakilian, and Nikos Zarifis. 2021. Learning Online Algorithms with Distributional Advice. In *ICML*, Vol. 139. 2687–2696.
[37] Yannis Dimopoulos, Vangelis Magirou, and Christos H. Papadimitriou. 1997. On Kernels, Defaults and Even Graphs. *Ann. Math. Artif. Intell.* 20, 1-4 (1997), 1–12.
[38] Michael Dinitz, Sungjin Im, Thomas Lavastida, Benjamin Moseley, and Sergei Vassilvitskii. 2022. Algorithms with Prediction Portfolios. In *NeurIPS*.
[39] Vibhor Dodeja, Mohammad Almasri, Rakesh Nagi, Jinjun Xiong, and Wen-Mei Hwu. 2022. PARSEC: PARallel Subgraph Enumeration in CUDA. In *IPDPS*. 168–178.
[40] Marina Drygala, Sai Ganesh Nagarajan, and Ola Svensson. 2023. Online Algorithms with Costly Predictions. In *AISTATS*, Vol. 206. 8078–8101.
[41] Keyu Duan, Zirui Liu, Peihao Wang, Wenqing Zheng, Kaixiong Zhou, Tianlong Chen, Xia Hu, and Zhangyang Wang. 2022. A Comprehensive Study on Large-Scale Graph Training: Benchmarking and Rethinking. In *NeurIPS*.
[42] Sourav Dutta, Pratik Nayek, and Arnab Bhattacharya. 2017. Neighbor-Aware Search for Approximate Labeled Graph Matching using the Chi-Square Statistics. In *WWW*. 1281–1290.
[43] Wenfei Fan, Yinghui Wu, and Jingbo Xu. 2016. Functional dependencies for graphs. In *SIGMOD*. ACM, 1843–1857.
[44] Jörg Flum and Martin Grohe. 2006. *Parameterized Complexity Theory*. Springer.
[45] Michael Garey and David Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company.
[46] Oded Goldreich. 2008. *Computational complexity - A conceptual perspective*. Cambridge University Press.
[47] Petr A. Golovach, Pinar Heggernes, Dieter Kratsch, and Yngve Villanger. 2015. An Incremental Polynomial Time Algorithm to Enumerate All Minimal Edge Dominating Sets. *Algorithmica* 72, 3 (2015), 836–859.
[48] Themistoklis Gouleakis, Konstantinos Lakis, and Golnoosh Shahkarami. 2023. Learning-Augmented Algorithms for

Online TSP on the Line. In *AAAI*. 11989–11996.
[49] Wentian Guo, Yuchen Li, Mo Sha, Bingsheng He, Xiaokui Xiao, and Kian-Lee Tan. 2020. GPU-Accelerated Subgraph Enumeration on Partitioned Graphs. In *SIGMOD*. 1067–1082.
[50] Wentian Guo, Yuchen Li, and Kian-Lee Tan. 2022. Exploiting Reuse for GPU Subgraph Enumeration. *TKDE* 34, 9 (2022), 4231–4244.
[51] Anupam Gupta, Debmalya Panigrahi, Bernardo Subercaseaux, and Kevin Sun. 2022. Augmenting Online Algorithms with ε-Accurate Predictions. In *NeurIPS*.
[52] William L. Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *NIPS*. 1024–1034.
[53] Myoungji Han, Hyunjoon Kim, Geonmo Gu, Kunsoo Park, and Wook-Shin Han. 2019. Efficient Subgraph Matching: Harmonizing Dynamic Programming, Adaptive Matching Order, and Failing Set Together. In *SIGMOD*. 1429–1446.
[54] Kongzhang Hao, Long Yuan, and Wenjie Zhang. 2021. Distributed Hop-Constrained s-t Simple Path Enumeration at Billion Scale. *PVLDB* 15, 2 (2021), 169–182.
[55] Jiezhong He, Zhouyang Liu, Yixin Chen, Hengyue Pan, Zhen Huang, and Dongsheng Li. 2022. FAST: A Scalable Subgraph Matching Framework over Large Graphs. In *HPEC*. 1–7.
[56] Lane A. Hemaspaandra and Mayur Thakur. 2004. Lower bounds and the hardness of counting properties. *Theor. Comput. Sci.* 326, 1-3 (2004), 1–28.
[57] Intel Corporation. 2013. *Intel® Advanced Vector Extensions 2 (Intel® AVX2)*. https://www.intel.com/content/www/us/en/docs/intrinsics-guide/index.html.
[58] Alexandru Iosup, Tim Hegeman, Wing Lung Ngai, Stijn Heldens, Arnau Prat-Pérez, Thomas Manhardt, Hassan Chafi, Mihai Capota, Narayanan Sundaram, Michael J. Anderson, Ilie Gabriel Tanase, Yinglong Xia, Lifeng Nai, and Peter A. Boncz. 2016. LDBC Graphalytics: A benchmark for large-scale graph analysis on parallel and distributed platforms. *PVLDB* 9, 13 (2016), 1317–1328.
[59] Guillaume Jaume, An-phi Nguyen, María Rodríguez Martínez, Jean-Philippe Thiran, and Maria Gabrani. 2019. edGNN: A Simple and Powerful GNN for Directed Labeled Graphs. *CoRR* abs/1904.08745 (2019).
[60] Zhihao Jia, Sina Lin, Mingyu Gao, Matei Zaharia, and Alex Aiken. 2020. Improving the Accuracy, Scalability, and Performance of Graph Neural Networks with Roc. In *MLSys*.
[61] Zhihao Jiang, Debmalya Panigrahi, and Kevin Sun. 2022. Online Algorithms for Weighted Paging with Predictions. *ACM Trans. Algorithms* 18, 4 (2022), 39:1–39:27.
[62] Zite Jiang, Shuai Zhang, Xingzhong Hou, Mengting Yuan, and Haihang You. 2024. IVE: Accelerating Enumeration-Based Subgraph Matching via Exploring Isolated Vertices. In *ICDE*. 4208–4221.
[63] Elias B. Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. 2017. Learning Combinatorial Optimization Algorithms over Graphs. In *NeurIPS*. 6348–6358.
[64] Hyunjoon Kim, Yunyoung Choi, Kunsoo Park, Xuemin Lin, Seok-Hee Hong, and Wook-Shin Han. 2021. Versatile Equivalences: Speeding up Subgraph Query Processing and Subgraph Matching. In *SIGMOD*. 925–937.
[65] Minsu Kim, Jinkyoo Park, and Joungho Kim. 2021. Learning Collaborative Policies to Solve NP-hard Routing Problems. In *NeurIPS*. 10418–10430.
[66] Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. 2018. The Case for Learned Index Structures. In *SIGMOD*. 489–504.
[67] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. 2005. Graphs over time: Densification laws, shrinking diameters and possible explanations. In *SIGKDD*.
[68] Ronghua Li, Sen Gao, Lu Qin, Guoren Wang, Weihua Yang, and Jeffrey Xu Yu. 2020. Ordering Heuristics for k-clique Listing. *PVLDB* 13, 11 (2020), 2536–2548.
[69] Zhiyuan Li, Xun Jian, Yue Wang, Yingxia Shao, and Lei Chen. 2024. DAHA: Accelerating GNN Training with Data and Hardware Aware Execution Planning. *PVLDB* 17, 6 (2024), 1364–1376.
[70] Wenqing Lin, Xiaokui Xiao, Xing Xie, and Xiaoli Li. 2017. Network Motif Discovery: A GPU Approach. *TKDE* 29, 3 (2017), 513–528.
[71] Xin Liu and Yangqiu Song. 2022. Graph Convolutional Networks with Dual Message Passing for Subgraph Isomorphism Counting and Matching. In *AAAI*. 7594–7602.
[72] Xuanzhou Liu, Lin Zhang, Jiaqi Sun, Yujiu Yang, and Haiqin Yang. 2023. D2Match: Leveraging Deep Learning and Degeneracy for Subgraph Matching. In *ICML*, Vol. 202. 22454–22472.
[73] Thodoris Lykouris and Sergei Vassilvitskii. 2018. Competitive caching with machine learned advice. *CoRR* abs/1802.05399 (2018). arXiv:1802.05399 http://arxiv.org/abs/1802.05399
[74] Lu Ma, Zeang Sheng, Xunkai Li, Xinyi Gao, Zhezheng Hao, Ling Yang, Xiaonan Nie, Jiawei Jiang, Wentao Zhang, and Bin Cui. 2025. Acceleration Algorithms in GNNs: A Survey. *TKDE* 37, 6 (2025), 3173–3192.
[75] Yining Ma, Jingwen Li, Zhiguang Cao, Wen Song, Le Zhang, Zhenghua Chen, and Jing Tang. 2021. Learning to Iteratively Solve Routing Problems with Dual-Aspect Collaborative Transformer. In *NeurIPS*. 11096–11107.

[76] Jessica Maghakian, Russell Lee, Mohammad Hajiesmaili, Jian Li, Ramesh K. Sitaraman, and Zhenhua Liu. 2023. Applied Online Algorithms with Heterogeneous Predictors. In *ICML*, Vol. 202. 23484–23497.
[77] Hamed Masnadi-Shirazi and Nuno Vasconcelos. 2015. A view of margin losses as regularizers of probability estimates. *J. Mach. Learn. Res.* 16 (2015), 2751–2795.
[78] Nikolai Merkel, Ruben Mayer, Tawkir Ahmed Fakir, and Hans-Arno Jacobsen. 2023. Partitioner Selection with EASE to Optimize Distributed Graph Processing. In *ICDE*. 2400–2414.
[79] Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Wenjie Jiang, Ebrahim M. Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Azade Nazi, Jiwoo Pak, Andy Tong, Kavya Srinivasa, William Hang, Emre Tuncer, Quoc V. Le, James Laudon, Richard Ho, Roger Carpenter, and Jeff Dean. 2021. A graph placement methodology for fast chip design. *Nature* 594, 7862 (2021), 207–212.
[80] Michael Mitzenmacher and Sergei Vassilvitskii. 2020. Algorithms with Predictions. In *Beyond the Worst-Case Analysis of Algorithms*. 646–662.
[81] Andrés Muñoz Medina and Sergei Vassilvitskii. 2017. Revenue Optimization with Approximate Bid Predictions. *CoRR* abs/1706.04732 (2017). arXiv:1706.04732 http://arxiv.org/abs/1706.04732
[82] Linh Anh Nguyen. 2018. Computing Bisimulation-Based Comparisons. *Fundam. Informaticae* 157, 4 (2018), 385–401.
[83] Iván Olmos, Jesús A. González, and Mauricio Osorio. 2007. Reductions between the Subgraph Isomorphism Problem and Hamiltonian and SAT Problems. In *CONIELECOMP*. 20.
[84] Martin Otto. 2020. Graded modal logic and counting bisimulation. *CoRR* arXiv:1910.00039 (2020). https://doi.org/10.48550/arXiv.1910.00039
[85] Christos H Papadimitriou. 2003. *Computational Complexity*. John Wiley and Sons Ltd.
[86] Manish Purohit, Zoya Svitkina, and Ravi Kumar. 2018. Improving Online Algorithms via ML Predictions. In *NeurIPS*. 9684–9693.
[87] Ruizhong Qiu, Zhiqing Sun, and Yiming Yang. 2022. DIMES: A Differentiable Meta Solver for Combinatorial Optimization Problems. In *NeurIPS*.
[88] Jiyuan Ren, Yangfu Liu, Yi Shen, Zhe Wang, and Zhen Luo. 2020. Approximate Sub-graph Matching over Knowledge Graph. In *MobiCASE*, Vol. 341. 198–208.
[89] Dhruv Rohatgi. 2020. Near-Optimal Bounds for Online Caching with Machine Learned Advice. In *SODA*. 1834–1845.
[90] Indradyumna Roy, Venkata Sai Baba Reddy Velugoti, Soumen Chakrabarti, and Abir De. 2022. Interpretable Neural Subgraph Matching for Graph Retrieval. In *AAAI*. 8115–8123.
[91] Michael Sejr Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. 2018. Modeling Relational Data with Graph Convolutional Networks. In *ESWC*, Vol. 10843. 593–607.
[92] Johannes Schmidt. 2009. *Enumeration: Algorithms and Complexity*. Ph. D. Dissertation. Universiteit Utrecht.
[93] Tim Schwabe and Maribel Acosta. 2024. Cardinality Estimation over Knowledge Graphs with Embeddings and Graph Neural Networks. *Proc. ACM Manag. Data* 2, 1 (2024), 44:1–44:26.
[94] Nino Shervashidze, Pascal Schweitzer, Erik Jan Van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. 2011. Weisfeiler-Lehman graph kernels. *Journal of Machine Learning Research* 12, 9 (2011).
[95] Zhichao Shi, Youhuan Li, Ziming Li, Yuequn Dou, Xionghu Zhong, and Lei Zou. 2025. MatCo: Computing Match Cover of Subgraph Query over Graph Data. *Proc. ACM Manag. Data* 3, 3 (2025), 186:1–186:24.
[96] Dave Steinkrau, Patrice Y. Simard, and Ian Buck. 2005. Using GPUs for Machine Learning Algorithms. In *ICDAR*. 1115–1119.
[97] Yann Strozecki. 2013. On Enumerating Monomials and Other Combinatorial Structures by Polynomial Interpolation. *TCS* 53, 4 (2013), 532–568.
[98] Yann Strozecki. 2013. On Enumerating Monomials and Other Combinatorial Structures by Polynomial Interpolation. *Theory Comput. Syst.* 53, 4 (2013), 532–568.
[99] Haoran Sun, Katayoon Goshvadi, Azade Nova, Dale Schuurmans, and Hanjun Dai. 2023. Revisiting Sampling for Combinatorial Optimization. In *ICML*, Vol. 202. 32859–32874.
[100] Shixuan Sun, Xibo Sun, Yulin Che, Qiong Luo, and Bingsheng He. 2020. RapidMatch: A Holistic Approach to Subgraph Query Processing. *PVLDB* 14, 2 (2020), 176–188.
[101] Xibo Sun and Qiong Luo. 2023. Efficient GPU-Accelerated Subgraph Matching. *Proc. ACM Manag. Data* 1, 2 (2023).
[102] Zhiqing Sun and Yiming Yang. 2023. DIFUSCO: Graph-based Diffusion Solvers for Combinatorial Optimization. In *NeurIPS*.
[103] Yujuan Tan, Zhuoxin Bai, Duo Liu, Zhaoyang Zeng, Yan Gan, Ao Ren, Xianzhang Chen, and Kan Zhong. 2024. BGS: Accelerate GNN training on multiple GPUs. *J. Syst. Archit.* 153 (2024), 103162.
[104] Yuya Tatsugi and Akira Nukada. 2022. Accelerating data transfer between host and device using idle GPU. In *GPGPU@PPoPP*. 2:1–2:6.
[105] Leslie G. Valiant. 1979. The Complexity of Enumeration and Reliability Problems. *SIAM J. Comput.* 8, 3 (1979), 410–421.

[106] Vladimir Vapnik. 1998. *Statistical learning theory.* Wiley.
[107] Luiz Felipe Vecchietti, Minah Seo, and Dongsoo Har. 2022. Sampling Rate Decay in Hindsight Experience Replay for Robot Control. *IEEE Trans. Cybern.* 52, 3 (2022), 1515–1526.
[108] Erik Vee, Sergei Vassilvitskii, and Jayavel Shanmugasundaram. 2010. Optimal online assignment with forecasts. In *ACM Conference on Electronic Commerce (EC)*. 109–118.
[109] Ivan Vendrov, Ryan Kiros, Sanja Fidler, and Raquel Urtasun. 2016. Order-Embeddings of Images and Language. In *ICLR*.
[110] Hanchen Wang, Ying Zhang, Lu Qin, Wei Wang, Wenjie Zhang, and Xuemin Lin. 2022. Reinforcement Learning Based Query Vertex Ordering Model for Subgraph Matching. In *ICDE*. 245–258.
[111] Yiyuan Wang, Chenghou Jin, Shaowei Cai, and Qingwei Lin. 2023. PathLAD+: An Improved Exact Algorithm for Subgraph Isomorphism Problem. In *IJCAI*.
[112] Bo Tang Wenzhe Hou, Xiang Zhao. 2025. OptMatch: An Effcient and Generic Neural Network-assisted Subgraph Matching Approach. In *ICDE*.
[113] Rajesh Wunnava and Taylor Riggan. 2020. Building a customer identity graph with Amazon Neptune. https://aws.amazon.com/blogs/database/building-a-customer-identity-graph-with-amazon-neptune/. Accessed: 2025-10-25.
[114] Chenyang Xu and Benjamin Moseley. 2022. Learning-Augmented Algorithms for Online Steiner Tree. In *AAAI*. 8744–8752.
[115] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How Powerful are Graph Neural Networks?. In *ICLR*. OpenReview.net.
[116] Xin Xue, Tianchen Zhu, Qingyun Sun, Haoyi Zhou, and Jianxin Li. 2025. Efficient Subgraph Matching Algorithm with Graph Neural Network. *Journal of Computer Research and Development*, 62, 3 (2025), 694–708.
[117] Haoran Ye, Jiarui Wang, Zhiguang Cao, Helan Liang, and Yong Li. 2023. DeepACO: Neural-enhanced Ant Systems for Combinatorial Optimization. In *NeurIPS*.
[118] Yutong Ye, Xiang Lian, and Mingsong Chen. 2024. Efficient Exact Subgraph Matching via GNN-based Path Dominance Embedding. *PVLDB* 17, 7 (2024), 1628–1641.
[119] Rex Ying, Tianyu Fu, Andrew Wang, Jiaxuan You, Yu Wang, and Jure Leskovec. 2024. Representation Learning for Frequent Subgraph Mining. *CoRR* abs/2402.14367 (2024). https://doi.org/10.48550/arXiv.2402.14367
[120] Rex Ying, Zhaoyu Lou, Jiaxuan You, Chengtao Wen, Arquimedes Canedo, and Jure Leskovec. 2020. Neural subgraph matching. *arXiv preprint arXiv:2007.03092*.
[121] Jiaxuan You, Jonathan Michael Gomes Selman, Rex Ying, and Jure Leskovec. 2021. Identity-aware Graph Neural Networks. In *AAAI*. 10737–10745.
[122] Yuanyuan Zeng, Yixiang Fang, Chenhao Ma, Xu Zhou, and Kenli Li. 2024. Efficient Distributed Hop-Constrained Path Enumeration on Large-Scale Graphs. *Proc. ACM Manag. Data* 2, 1 (2024), 22:1–22:25.
[123] Cong Zhang, Wen Song, Zhiguang Cao, Jie Zhang, Puay Siew Tan, and Chi Xu. 2020. Learning to Dispatch for Job Shop Scheduling via Deep Reinforcement Learning. In *NeurIPS*.
[124] Dinghuai Zhang, Hanjun Dai, Nikolay Malkin, Aaron C Courville, Yoshua Bengio, and Ling Pan. 2023. Let the flows tell: Solving graph combinatorial problems with gflownets. *Advances in neural information processing systems* 36 (2023), 11952–11969.
[125] Lizhi Zhang, Zhiquan Lai, Shengwei Li, Yu Tang, Feng Liu, and Dongsheng Li. 2021. 2PGraph: Accelerating GNN Training over Large Graphs on GPU Clusters. In *CLUSTER*. 103–113.
[126] Shijie Zhang, Jiong Yang, and Wei Jin. 2010. SAPPER: Subgraph Indexing and Approximate Matching in Large Graphs. *PVLDB* 3, 1 (2010), 1185–1194.
[127] Yuhao Zhang and Arun Kumar. 2023. Lotan: Bridging the Gap between GNNs and Scalable Graph Analytics Engines. *PVLDB* 16, 11 (2023), 2728–2741.
[128] Zhijie Zhang, Yujie Lu, Weiguo Zheng, and Xuemin Lin. 2024. A Comprehensive Survey and Experimental Study of Subgraph Matching: Trends, Unbiasedness, and Interaction. *Proc. ACM Manag. Data* 2, 1 (2024), 60:1–60:29.
[129] Kangfei Zhao, Jeffrey Xu Yu, Hao Zhang, Qiyan Li, and Yu Rong. 2021. A Learned Sketch for Subgraph Counting. In *SIGMOD*. 2142–2155.
[130] Yu Zhao, Chunhong Zhang, Tingting Sun, Yang Ji, Zheng Hu, and Xiaofeng Qiu. 2016. Approximate Subgraph Matching Query over Large Graph. In *BigCom*, Vol. 9784. 247–256.

Received July 2025; revised October 2025; accepted November 2025