LIVE: Learnable Monotonic Vertex Embedding for Efficient Exact Subgraph Matching

Abstract

Exact subgraph matching is a fundamental graph operator that supports many graph analytics tasks, yet it remains computationally challenging due to its NP-completeness. Recent learning-based approaches accelerate query processing via dominance-preserving vertex embeddings, but they suffer from expensive offline training, limited pruning effectiveness, and heavy reliance on complex index structures, all of which hinder the scalability to large graphs. In this paper, we propose L̲earnable MonotonI̲c V̲ertex E̲mbedding (LIVE), a learning-based framework for efficient exact subgraph matching that scales to large graphs. LIVE enforces monotonicity among vertex embeddings by design, making dominance correctness an inherent structural property and enabling embedding learning to directly optimize vertex-level pruning power. To this end, we introduce a query cost model with a differentiable surrogate objective to guide efficient offline training. Moreover, we design a lightweight one-dimensional *iLabel* index that preserves dominance relationships and supports efficient online query processing. Extensive experiments on both synthetic and real-world datasets demonstrate that LIVE significantly outperforms state-of-the-art methods in efficiency and pruning effectiveness.

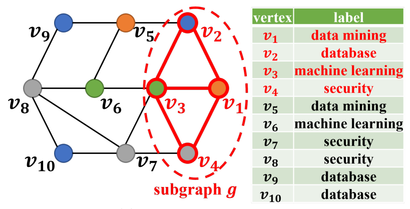
1 Introduction

Graph data management has become increasingly important in a wide range of real-world applications, *e.g.,* social network analysis [2], knowledge graph discovery [35], and biological network mining [50]. Among various graph operators, exact subgraph matching is one of the most fundamental yet computationally challenging tasks. Given a large data graph and a user-specified query graph, an exact subgraph matching query aims to retrieve all subgraphs that are isomorphic to the query graph, preserving both structural and label consistency. Such a query plays a central role in many graph data-driven analytical tasks, *e.g.,* pattern recognition [57], community search [63], and graph-based query answering [16].

Below, we give an example of a subgraph matching query for the discovery of research collaboration patterns in academic networks.

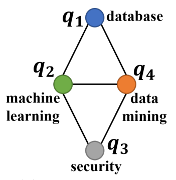
Example 1. ***(Research Collaboration Pattern Discovery in Academic Networks)*** *As shown in Figure 1(a), an academic collaboration network can be modeled as an undirected, vertex-labeled graph, where each vertex represents a researcher labeled by their primary research field* (e.g., *database, data mining, machine learning, and security), and each edge represents a collaborative relationship among two researchers. Such networks naturally form large-scale graphs capturing scholarly collaborations.*

A common analytical task is to identify recurring collaboration structures that match a particular team formation pattern. For example, a conference organizer may wish to find research teams exhibiting a specific interdisciplinary pattern, e.g., *"a core researcher in data mining collaborating with colleagues in database systems, machine learning, security". This pattern can be expressed as a query graph q (Figure 1(b)), where vertices denote researchers with specified research fields and edges denote collaborations.*



vertex	label
v_1	data mining
v_2	database
v_3	machine learning
v_4	security
v_5	data mining
v_6	machine learning
v_7	security
v_8	security
v_9	database
v_{10}	database

(a) academic network G



(b) query graph q

Figure 1: An example of the exact subgraph matching.

Given a large academic network G (Figure 1(a)), exact subgraph matching retrieves all subgraphs $g \subseteq G$ isomorphic to the query graph q. As illustrated, one valid match is $g = \{v_1, v_2, v_3, v_4\}$, corresponding to researchers in data mining, database, machine learning, and security, with all collaborations preserved. □

Despite its importance, exact subgraph matching remains highly challenging. The problem has been proven to be NP-complete [15, 19], and the search space grows exponentially with graph size and structural complexity, *e.g.,* an exact subgraph matching query on a graph with only $3K$ vertices and $10K$ edges can yield over 4 billion answers [47], rendering such queries very costly on large graphs.

Prior Arts. Existing subgraph matching methods can be broadly classified into the following two categories.

(1) Structural-based approaches [9–11, 22, 24, 26, 45, 47]. These methods rely on explicit structural comparisons between the query graph and candidate subgraphs, which often incur high computational overhead and limit query efficiency.

(2) Learning-based approaches [8, 33, 36, 52, 59, 60]. Recent works in this line leverage Graph Neural Networks (GNNs) to embed graphs into embedding spaces, transforming subgraph matching into efficient vector-based operations. Early studies primarily targeted approximate matching or graph similarity search [8, 33, 36], where embeddings are used to estimate structural similarity rather than to guarantee exact matches. More recent efforts have extended learning-based techniques to exact subgraph matching, either by learning matching orders [52, 59] or by preserving dominance relationships in the embedding space [60].

Among them, dominance-based methods (*e.g.,* GNN-PE [60]) learn vertex embeddings from labels and 1-hop neighborhoods, such that subgraph containment can be verified via coordinate-wise dominance. This property enables candidate filtering without false dismissals and yields substantial speedups over traditional structure-based methods (*e.g.,* up to 1–2 orders of magnitude [60]).

Despite these advances, dominance-learning–based approaches still suffer from two fundamental limitations: (i) expensive offline training that requires enumerating 1-hop subgraphs and all possible 1-hop substructures for each vertex, and (ii) limited pruning effectiveness during online queries, as embedding learning is tightly constrained by dominance correctness. To compensate for the latter, existing methods often rely on complex index structures [60], which incur substantial storage and traversal overhead.

Consequently, while learning-based approaches make exact subgraph matching feasible on small to medium-sized graphs, they remain inadequate for large-scale graphs. For instance, these methods often fail to complete offline preprocessing on graphs with millions of vertices and edges within practical time budgets [60]. This leads to a natural question: *Is it possible to design a learning-based approach that further accelerates exact subgraph matching with a lightweight offline training phase, while enabling efficient online query processing without relying on heavy index structures?*

Our Contributions. In this paper, we answer the question above affirmatively by proposing *Learnable MonotonIc Vertex Embedding* (LIVE), a novel framework for efficient exact subgraph matching on large graphs. Unlike prior learning-based approaches, LIVE rethinks the coupling between no-false-dismissal guarantees and pruning power for vertex embedding learning in exact subgraph matching.

Specifically, LIVE introduces a design that enforces monotonicity among vertex embeddings by construction. As a result, the no-false-dismissal guarantee of candidate filtering becomes an inherent structural property of the embeddings, rather than an outcome learned through enumerating subgraph–substructure pairs. This decoupling allows embedding learning to move beyond mere correctness preservation and directly optimize embedding quality, *i.e.,* vertex-level pruning power. Accordingly, we develop a cost model that estimates the query cost induced by vertex embeddings, and derive a continuously differentiable surrogate of this model, enabling effective optimization of vertex embeddings during offline training and leading to more efficient online query processing.

Building on the learned monotonic vertex embeddings, we further design a lightweight *iLabel* index that maps multi-dimensional embeddings into a one-dimensional key space while preserving dominance relationships. This design transforms dominance-region search for query vertices to efficient range queries supported by a B^+-tree, thereby eliminating the need for complex indexing structures. In addition, we incorporate multiple pruning strategies during index traversal to further reduce the candidate search space.

In summary, we make the following contributions:

- We propose LIVE (Section 2), a learning-based framework for efficient exact subgraph matching scaling to large graphs.
- We develop a learnable monotonic vertex embedding method (Section 3) that enforces dominance relationships by design, allowing embedding learning to focus solely on optimizing vertex-level pruning power. We further introduce a query cost model and derive a continuous, differentiable surrogate objective to guide offline training.
- We design a lightweight one-dimensional *iLabel* index (Section 4) that preserves dominance relationships among multi-dimensional vertex embeddings, enabling efficient online query processing with low storage and traversal overhead.
- We propose an efficient online query processing algorithm (Section 5) that integrates multiple pruning strategies supported by the *iLabel* index.
- We conduct extensive experiments on both synthetic and real-world datasets (Section 6), confirming the efficiency and effectiveness of our LIVE for exact subgraph matching.

We review related work in Section 7 and conclude in Section 8.

2 Problem Definition

This section formally defines the graph data model and the subgraph matching problem over a graph database. Table 1 summarizes the frequently used symbols and their descriptions.

Table 1: Symbols and Descriptions

Symbol	Description
G	a data graph
q	a query graph
g	a subgraph of the data graph G
v_i (or q_i)	a vertex in graph G (or q)
e_{ij} (or $e_{q_iq_j}$)	an edge in graph G (or q)
$V(G)$ (or $V(q)$)	a set of vertices v_i (or q_i) in graph G (or q)
$L(G)$ (or $L(q)$)	a labeling function of graph G (or q)
$o(v_i)$ (or $o(q_i)$)	a vertex embedding vector of v_i (or q_i)
$g_1(v_i)$ (or $s_1(v_i)$)	a 1-hop subgraph (or substructure) of v_i
$\mathcal{N}_1(v_i)$ (or $\mathcal{N}_1(q_i)$)	a set of v_i's (or q_i's) 1-hop neighbors

2.1 Preliminaries

We start with basic notations.

Graph. A graph, G is denoted by $G = (V(G), E(G), \phi(G), L(G))$, where $V(G)$ is a set of vertices v_i, $E(G)$ is a set of edges $e_{ij} = (v_i, v_j)$ connecting vertex pairs, $\phi(G)$ is a mapping function from vertex pairs to edges (*i.e.,* $\Phi(G){:}V(G) \times V(G) \rightarrow E(G)$), and $L(G)$ is a vertex labeling function that assigns each vertex $v_i \in V(G)$ a label $L(v_i)$ describing its attribute or type.

In this paper, we consider an undirected labeled graph, which is widely used in applications such as social networks [53], knowledge graphs [35], and biological interaction networks [28], where vertices represent entities and edges capture their relationships.

Graph Isomorphism [7, 20]. Given two graphs $G_1 = (V_1, E_1, \phi_1, L_1)$ and $G_2 = (V_2, E_2, \phi_2, L_2)$, G_1 is said to be isomorphic to G_2, denoted by $G_1 \equiv G_2$, if there exists a bijection mapping $f : V_1 \rightarrow V_2$ satisfying the following two conditions: (i) *label preservation*: for each vertex $v_i \in V_1$, $L_1(v_i) = L_2(f(v_i))$; and (ii) *edge preservation*: for any vertex pair $(v_i, v_j) \in E_1$, $(f(v_i), f(v_j)) \in E_2$.

Intuitively, G_1 and G_2 are isomorphic if they have identical structural topology and vertex labels under the mapping f.

Subgraph Isomorphism. Let G be a data graph and q be a query graph. We say that q is subgraph-isomorphic to G, denoted by $q \subseteq G$, if there exists a subgraph $g \subseteq G$ such that $g \equiv q$.

The subgraph isomorphism problem asks whether a query graph q exactly matches a subgraph of G while preserving structure and labels. The problem has been proven to be NP-complete [15, 19].

2.2 Exact Subgraph Matching Query

Based on the preliminaries in Section 2.1, we formally define the subgraph matching query as follows:

Definition 1. ***(Exact Subgraph Matching Query)*** *Given a large data graph G and a query graph q, an exact subgraph matching query retrieves all subgraphs $g \subseteq G$ such that $g \equiv q$.* □

In other words, the goal is to enumerate all subgraphs of G that are structurally identical to the query pattern q. Exact subgraph matching is fundamental to many applications, including pattern discovery in social networks [42], molecule search in bioinformatics [3], and entity-relation extraction in knowledge graphs [44].

2.3 Challenges

Recent learning-based techniques [52, 59] have explored exact subgraph matching via learned matching orders, while dominance-learning-based methods [60] further enable embedding-based candidate filtering without false dismissals. However, integrating learning into exact subgraph matching poses several challenges.

First, to guarantee correctness during candidate search, existing methods often require enumerating a massive number of subgraph–substructure pairs to learn dominance-preserving embeddings, resulting in substantial offline training costs and limited scalability. Second, prior embedding learning approaches are primarily driven by correctness constraints and do not explicitly optimize the pruning power of vertex embeddings, leading to suboptimal filtering effectiveness during query processing, especially on large graphs. Finally, even with dominance-preserving embeddings, efficiently retrieving all dominating vertices typically relies on complex indexing structures, incurring high storage and traversal overhead.

These challenges highlight the need for a new framework that preserves dominance correctness while scaling to large graphs in both offline vertex embedding training and online query processing.

2.4 The LIVE Framework

Algorithm 1 illustrates a novel Learning-based Vertex Embedding (LIVE) framework for efficiently answering exact subgraph matching queries on large graphs using learned vertex embeddings. The framework consists of two phases: an offline precomputation phase (lines 1–3) and an online subgraph matching phase (lines 4–9).

Offline Phase. The offline phase learns vertex embeddings and builds an index to support efficient query processing. Specifically, LIVE trains an embedding model Emb(·) on the data graph G using a proposed cost model (see Section 3.3) to minimize the expected query cost induced by vertex embeddings (line 1). After training, LIVE generates an embedding $o(v_i)$ for each data vertex $v_i \in V(G)$ using the learned model (line 2). Based on these embeddings, an *iLabel* index $\mathcal{I}$ is constructed by mapping multi-dimensional vertex embeddings into a one-dimensional key space, enabling efficient range-based retrieval during query processing (line 3).

Online Phase. Given a query graph q, the online phase retrieves and refines candidate vertices to enumerate exact matches. For each query vertex $q_i \in V(q)$, LIVE first computes its embedding $o(q_i)$ using the trained model (lines 4–5). The *iLabel* index $\mathcal{I}$ is then traversed to obtain a candidate set $q_i.cand\_set$ of vertices in G that may match q_i (line 6). After candidate sets are generated for all query vertices, LIVE determines a matching order Q of query vertices $q_i \in q$ based on their candidate set sizes, prioritizing query vertices with fewer candidates (line 7). Finally, an exact refinement procedure assembles and verifies candidate subgraphs by extending partial matches according to the matching order and enforcing subgraph isomorphism constraints (line 8). Finally, LIVE returns all subgraphs $g \subseteq G$ that are isomorphic to q (line 9).

The design of LIVE is detailed in the following sections. Sections 3 and 4 describe the offline phase, including monotonic vertex embedding learning and *iLabel* index construction, respectively, while Section 5 presents the online subgraph matching algorithm.

Note. The key advantage of LIVE lies in its principled separation of correctness guarantees from efficiency-oriented optimization. By enforcing dominance correctness as an intrinsic property of vertex embeddings through monotonicity, LIVE allows embedding learning to directly optimize vertex-level pruning power for improved query efficiency. Meanwhile, the dominance-preserving index reduces the cost of candidate retrieval while ensuring exactness, *i.e.,* no false dismissals occur during index traversal.

Algorithm 1: The LIVE Framework for Exact Subgraph Matching

```
  Input: a data graph $G$ and a query graph $q$
  Output: subgraphs $g$ ($\subseteq G$) that are isomorphic to $q$
  // Offline Pre-Computation Phase
  // Generate vertex embeddings
1 train a vertex embedding model Emb(·) that minimizes query processing
    cost over the data graph $G$
2 generate an embedding $o(v_i)$ for $\forall v_i \in G$ using the trained Emb(·)
  // Index construction
3 build an iLabel index $\mathcal{I}$ over the learned data vertex embeddings $o(v_i)$
  // Online Subgraph Matching Phase
4 for each query vertex $q_i \in q$ do
      // Retrieve candidate matching vertices
5     generate the query vertex embedding $o(q_i)$ using the trained
        embedding model Emb(·)
6     obtain the candidate matching vertices $q_i.cand\_set$ for $q_i$ by traversing
        the index $\mathcal{I}$
  // Obtain and refine candidate subgraphs
7 obtain a matching order $Q$ of query vertices $q_i \in q$ based on the sizes of the
    their candidate sets $|q_i.cand\_set|$
8 assemble and refine candidate subgraphs $g$ from candidate vertices in
    $q_i.cand\_set$
9 return subgraphs $g$ ($\equiv q$)
```

3 Learnable Monotonic Vertex Embedding

This section presents the design of monotonic vertex embeddings and the training of the embedding model Emb(·) in LIVE (lines 1–2 of Algorithm 1). Specifically, Section 3.1 introduces structural notions for characterizing vertex neighborhoods, Section 3.2 presents the monotonic embedding design and its dominance-preserving properties, and Section 3.3 describes cost-model-based embedding learning for achieving strong pruning power. **Due to space limitations, we defer all lemma proofs to Appendix A in [5].**

3.1 Preliminaries

We introduce two structural notions used throughout the paper, which serve as conceptual units for characterizing local matching feasibility without requiring explicit enumeration during learning.

1-hop Subgraph. Given a data graph G, the 1-hop subgraph centered at a vertex $v_i \in G$ is the induced subgraph containing v_i and all its immediate neighbors: $g_1(v_i) = G[\{v_i\} \cup \mathcal{N}_1(v_i)]$, where $\mathcal{N}_1(v_i)$ denotes the set of 1-hop neighbors of v_i in G. The 1-hop subgraph captures both the label of the center vertex v_i and the structural context of its local neighborhood.

In subgraph matching, $g_1(v_i)$ provides the necessary local context for determining whether $v_i \in G$ can serve as a candidate match for a query vertex $q_i \in q$. In particular, any valid match between q_i and v_i must satisfy local consistency constraints within their respective 1-hop neighborhoods.

1-hop Substructure. Given the 1-hop subgraph $g_1(v_i)$ of a data vertex $v_i \in G$, a 1-hop substructure $s_1(v_i)$ is a subgraph induced by v_i and a subset of its 1-hop neighbors $\mathcal{N}_1(v_i)$. For a query vertex $q_i \in q$ with 1-hop subgraph $g_1(q_i)$, if q_i can be matched to v_i, then there exists a substructure $s_1(v_i) \subseteq g_1(v_i)$ such that $g_1(q_i) \equiv s_1(v_i)$.

This observation yields a necessary condition for vertex-level matching: the local neighborhood $\mathcal{N}_1(q_i)$ of a query vertex $q_i \in q$ must be realizable as a substructure of the 1-hop subgraph of data vertex $v_i \in G$. Figure 2(a) illustrates an example 1-hop subgraph $g_1(v_1)$, and Figure 2(b) shows all 8 ($= 2^3$) possible 1-hop substructures contained in $g_1(v_1)$.

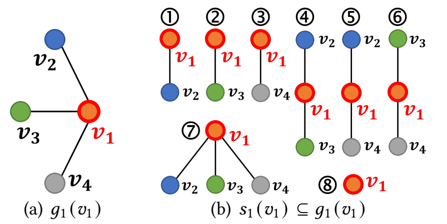


Figure 2: $g_1(v_1)$ and its all possible $s_1(v_1)$.

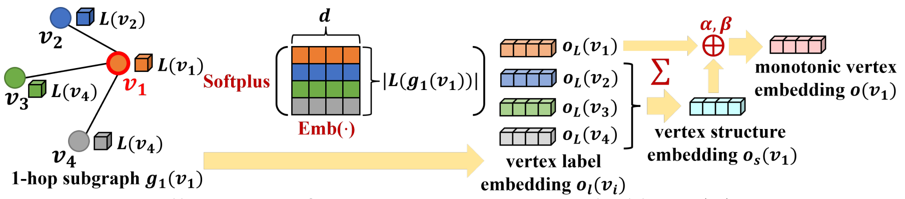


Figure 3: Illustration of our monotonic vertex embedding $o(v_i)$.

3.2 Monotonic Vertex Embedding Design

In this subsection, we present a vertex embedding design that maps each vertex and its local neighborhood into an embedding space, enabling efficient and exact candidate filtering via vertex embeddings. Under this design, subgraph inclusion relationships are preserved by construction, rather than learned from enumerated subgraph–substructure pairs as in [60].

Monotonic Vertex Embedding Function. We define a vertex embedding function $f : g_1(v_i) \to o(v_i) \in \mathbb{R}^d$, where $g_1(v_i)$ denotes the 1-hop subgraph centered at v_i, and $o(v_i)$ is the d-dimensional embedding of vertex v_i.

The embedding function $f(\cdot)$ is said to be *monotonic* with respect to subgraph inclusion if $s_1(v_i) \subseteq g_1(v_i) \implies o(s_1(v_i)) \preceq o(g_1(v_i))$, where $\preceq$ denotes *coordinate-wise dominance* between two embeddings, *i.e.*, $o(s_1(v_i))[j] \le o(g_1(v_i))[j]$ holds for the embedding values in all d dimensions [60, 61]. This monotonicity ensures that subgraph inclusion for each vertex in graph G is faithfully reflected as dominance in the learned embedding space.

Below, we show how the embedding function $f(\cdot)$ is constructed.

Embedding Design. The embedding $o(v_i)$ of a vertex v_i consists of two components: a vertex label embedding (VLE) that captures the semantic information of v_i, and a vertex structure embedding (VSE) that summarizes its local neighborhood structure.

Vertex Label Embedding (VLE). Let $o_l(v_i)$ denote the vertex label embedding of vertex v_i, defined as

$$o_l(v_i) = \text{softplus}(\text{Emb}(L(v_i))), \tag{1}$$

where Emb$(\cdot)$ is a trainable embedding model (see Section 3.3 for its training) that maps the vertex label $L(v_i)$ to a d-dimensional vector, and softplus$(x) = \log(1 + e^x)$. The softplus activation ensures that all embedding dimensions are strictly non-negative, which is required to preserve dominance relationships among vertices.

As illustrated in Figure 3, vertex label embeddings $o_l(v_1)$–$o_l(v_4)$ are generated for vertices v_1–v_4 according to their respective labels $L(v_1)$–$L(v_4)$ using Eq. 1.

Vertex Structure Embedding (VSE). Let $o_s(v_i)$ denote the vertex structure embedding, which captures the local structural context of v_i by aggregating the label embeddings of its 1-hop neighbors:

$$o_s(v_i) = \sum_{v_j \in \mathcal{N}_1(v_i)} o_l(v_j). \tag{2}$$

This aggregation summarizes the neighborhood structure of v_i in a manner consistent with its 1-hop subgraph $g_1(v_i)$. Since the aggregation is a summation over non-negative vectors, it is monotonic *w.r.t.* neighbor inclusion. For example, in Figure 3, the structure embedding $o_s(v_i)$ of vertex v_1 is obtained by summing the label embeddings of its neighbors, *i.e.*, $o_s(v_1) = o_l(v_2) + o_l(v_3) + o_l(v_4)$.

Monotonic Vertex Embedding. The final monotonic embedding of vertex v_i is a weighted combination of its VLE and VSE components:

$$o(v_i) = \alpha\, o_l(v_i) + \beta\, o_s(v_i), \tag{3}$$

where α and β are non-negative weighting parameters (see Section 3.3 for their settings). This formulation jointly encodes vertex semantics and local structural information while preserving monotonicity. As illustrated in Figure 3, the embedding of vertex v_1 is computed as $o(v_1) = \alpha\, o_l(v_1) + \beta\, o_s(v_1)$.

Monotonicity and Dominance Preservation. By construction, all components of $o_l(v_i)$ and $o_s(v_i)$ are non-negative, and both summation and weighted addition are monotonic operations. Consequently, for any 1-hop subgraph $g_1(v_i)$ and its 1-hop substructure $s_1(v_i)$ with $s_1(v_i) \subseteq g_1(v_i)$, their embeddings satisfy $o(s_1(v_i))[j] \le o(g_1(v_i))[j]$ for all $j \in [1, d]$. That is, the embedding function $f(\cdot)$ preserves inclusion relationships among 1-hop subgraphs as dominance relationships in the embedding space.

Lemma 1. ***(Monotonicity of Vertex Embeddings).*** *Given a 1-hop subgraph $g_1(v_i)$ and one of its 1-hop substructures $s_1(v_i)$ with $s_1(v_i) \subseteq g_1(v_i)$, their embeddings satisfy $o(s_1(v_i)) \preceq o(g_1(v_i))$.* □

No-False-Dismissal Guarantee. The monotonicity property directly provides a no-false-dismissal guarantee for vertex-level candidate filtering. If a query vertex $q_i \in q$ can be matched to a data vertex $v_i \in G$, then the 1-hop subgraph of q_i must be a substructure of $g_1(v_i)$. By the monotonic embedding design, their embeddings satisfy $o(q_i) \preceq o(v_i)$. Consequently, filtering candidates based on the dominance relationship among vertex embeddings preserves all valid matches and does not eliminate any true matches.

Example 2. *Figure 4 illustrates how monotonic vertex embedding enables safe and effective candidate filtering. In the data graph G, the 1-hop subgraph of vertex v_2 contains that of the query vertex q_1,* i.e., *$g_1(q_1) \subseteq g_1(v_2)$. By the monotonic embedding design, this inclusion relationship is preserved in the embedding space as a dominance relation,* i.e., *$o(q_1) \preceq o(v_2)$. Consequently, v_2 lies in the dominating region $DR(o(q_1))$, which consists of vertices whose embedding values are no smaller than $o(q_1)$ in all dimensions, and is correctly retained as a candidate for q_1 during index-based retrieval.*

Vertices whose embeddings fall outside $DR(o(q_1))$ violate the dominance condition and can be safely pruned without affecting correctness. Note that some vertices (e.g., *v_3*) *may satisfy the dominance condition yet fail to match q_1 due to higher-order structural constraints, which will be eliminated later (Section 5).* □

Decoupling Correctness Guarantees from Embedding Optimization. Unlike existing dominance learning-based exact subgraph matching methods (*e.g.*, [60]) that tightly couple the no-false-dismissal guarantee with the embedding learning objective, LIVE preserves subgraph containment as dominance relations in the embedding space by design, rather than through learning. As a result, the no-false-dismissal guarantee becomes an inherent structural property of the embedding, independent of the learning objective.

This decoupling enables a new optimization perspective. Once correctness is ensured by construction, embedding learning is freed from dominance constraints and can directly target pruning power

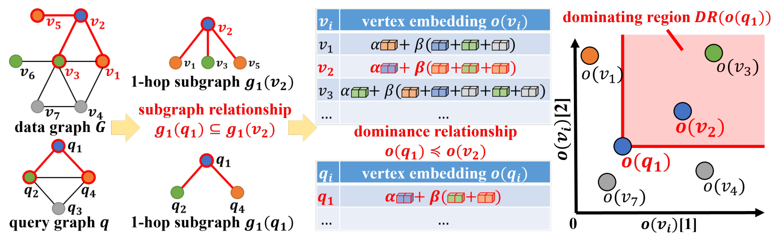
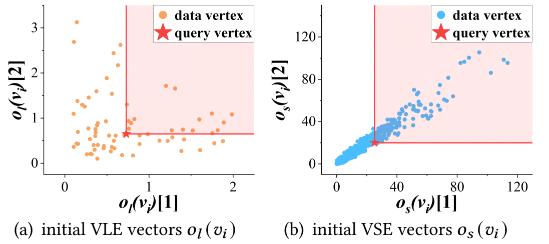

data graph G
query graph q
1-hop subgraph $g_1(v_2)$
1-hop subgraph $g_1(q_1)$
subgraph relationship $g_1(q_1) \subseteq g_1(v_2)$
dominance relationship $o(q_1) \preccurlyeq o(v_2)$
vertex embedding $o(v_i)$
vertex embedding $o(q_i)$
dominating region $DR(o(q_1))$
$o(v_i)[1]$
$o(v_i)[2]$

Figure 4: An illustration of the monotonicity of our vertex embedding.



(a) initial VLE vectors $o_l(v_i)$ (b) initial VSE vectors $o_s(v_i)$

Figure 5: An example of initial VLE/VSE distributions.

and query efficiency. This insight motivates the cost-model–driven embedding optimization introduced in Section 3.3, which explicitly minimizes query cost rather than learning dominance relations.

3.3 Cost Model-based Embedding Optimization

Given that the monotonic vertex embedding design in Section 3.2 guarantees dominance correctness by construction, in this subsection, we focus on improving embedding quality by maximizing pruning power to reduce query cost during candidate retrieval.

During online query processing, the cost of matching a query vertex q_i is determined by the number of data vertices whose embeddings are dominated by $o(q_i)$, *i.e.,* the size of the dominance-induced candidate set $\{v_j \mid o(q_i) \preceq o(v_j)\}$. Accordingly, embedding optimization aims to reduce the number of dominated vertices, thereby minimizing the size of candidates examined during query processing. However, query embeddings are unknown during offline training, and enumerating all possible 1-hop substructures is infeasible due to their exponential growth with vertex degree ($\sum_{v \in V(G)} 2^{deg(v)}$). To address this, we estimate query cost on data vertices and use it as a surrogate for expected query behavior.

Cost Model for Query Efficiency. Based on the above observation, we define a cost model that captures query efficiency by measuring the average number of vertices dominated by a data vertex embedding. The expected query cost is approximated as

$$\text{Cost} = \frac{1}{|V(G)|} \sum_{v_i \in V(G)} \left|\left\{v_j \in V(G) \mid o(v_i) \preceq o(v_j)\right\}\right|, \tag{4}$$

which represents the expected size of the candidate set induced by dominance-based filtering and serves as a proxy for query efficiency.

Figure 5 visualizes the initial distributions of 2D vertex label embeddings (VLE) and vertex structure embeddings (VSE) on the Yeast dataset. The VLE vectors are randomly initialized and thus uniformly scattered in the embedding space (Figure 5(a)), resulting in relatively small candidate sets. In contrast, VSE vectors, computed by summing non-negative VLE vectors, exhibit a strong concentration along the diagonal (Figure 5(b)). This concentration causes many vertices to satisfy the dominance condition, leading to large candidate sets and high query cost.

Anti-Dominance Objective and Continuous Relaxation. As shown in Eq. 4, the query cost induced by a query vertex embedding is determined by the number of data vertices whose embeddings are dominated by it. Accordingly, minimizing query cost is equivalent to minimizing the total number of dominance relations among vertex embeddings in the embedding space.

Intuitively, the notion of *anti-dominance*, which indicates an object dominates or is dominated by only a few others [12], naturally aligns with this optimization objective. Following this intuition, we formalize Eq. 4 using the following *anti-dominance cost*:

$$L_{\text{cost}} = \frac{1}{|V(G)|^2} \sum_{v_i \in V(G)} \sum_{v_j \in V(G)} \mathbf{1}\{o(v_i) \preceq o(v_j)\}, \tag{5}$$

where $\mathbf{1}\{\cdot\}$ is an indicator function that equals 1 if the embedding $o(v_i)$ dominates $o(v_j)$, and 0 otherwise. This objective measures the expected number of dominance relations among all vertex pairs; minimizing it directly reduces the number of dominated vertices and thus improves vertex-level pruning power during query processing.

However, the anti-dominance cost in Eq. 5 is inherently discrete due to the indicator function induced by coordinate-wise dominance, and therefore cannot be optimized directly using gradient-based methods. To enable efficient optimization, we derive a continuous and differentiable relaxation of this objective. Specifically, we approximate the indicator function with a sigmoid-based surrogate and define the *anti-dominance loss* as

$$L = \frac{1}{|V(G)|^2} \sum_{v_i \in V(G)} \sum_{v_j \in V(G)} \sigma\left(\frac{\min_k \left(o(v_i)[k] - o(v_j)[k]\right)}{\tau}\right), \tag{6}$$

where $\sigma(x) = \frac{1}{1+e^{-x}}$ is the sigmoid function and τ is a temperature parameter that controls the smoothness of the approximation. Note that the $\min_k(\cdot)$ operator is piecewise linear and subdifferentiable. Following PointNet [41] and Deep Sets [62], we use standard automatic differentiation [40] to obtain a valid subgradient; nondifferentiable ties occur with probability 0 for continuous parameters.

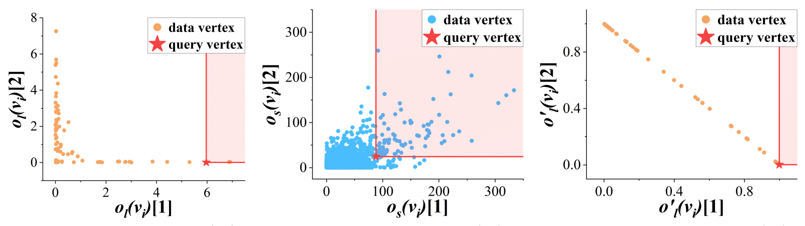
Lemma 2. ***(Relationship Between Anti-Dominance Loss and Discrete Objective).*** *Let L_{cost} and L be defined in Eqs. 5 and 6, respectively. As the temperature parameter $\tau \to 0^+$, the anti-dominance loss L converges to the discrete anti-dominance cost L_{cost}. Moreover, for any finite $\tau > 0$, L serves as a smooth upper bound of L_{cost}.* □

Although the anti-dominance loss in Eq. 6 provides a differentiable surrogate for the discrete cost objective, directly optimizing it requires evaluating all vertex pairs, incurring $O(|V(G)|^2)$ time/space cost and rendering it infeasible for large graphs. To address this, we adopt a sampling-based approximation during training: instead of summing over all vertex pairs, we uniformly sample vertex pairs (v_i, v_j) from $V(G) \times V(G)$ and optimize the expected anti-dominance loss:

$$L = \mathbb{E}_{(v_i,v_j)\sim\mathcal{U}(V(G)\times V(G))}\left[\sigma\left(\frac{\min_k \left(o(v_i)[k] - o(v_j)[k]\right)}{\tau}\right)\right] \tag{7}$$

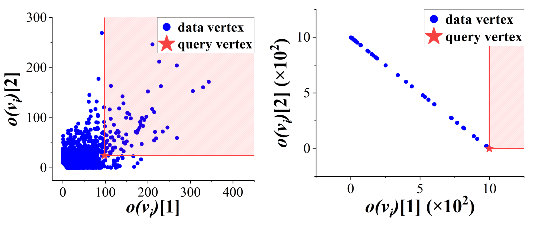
This sampling strategy preserves the expectation of the full loss in Eq. 6 while reducing the computational cost to be linear in the number of sampled pairs. Consequently, the anti-dominance loss can be efficiently optimized using stochastic gradient-based methods.

Model Training. We train the vertex embedding model Emb($\cdot$) by minimizing the sampled anti-dominance loss in Eq. 7 using mini-batch stochastic gradient descent. In each iteration, a mini-batch of vertex pairs is sampled to estimate gradients, while the temperature parameter τ is gradually annealed to tighten the approximation to



(a) learned VLE vectors $o_l(v_i)$ (b) learned VSE vectors $o_s(v_i)$ (c) optimized VLE vectors $o_l(v_i)$

Figure 6: An example of optimized VLE/VSE vector distributions.



(a) $\alpha = 10, \beta = 1$ (b) $\alpha = 1,000, \beta = 0.01$

Figure 7: An example of $o(v_i)$ with different α/β.

the discrete cost objective. After convergence, the trained model Emb($\cdot$) is used to generate embeddings for all data vertices in G.

Figures 6(a) and 6(b) show the distributions of VLE and VSE embeddings on the Yeast dataset after training. Compared to the initial distributions in Figure 5, both embeddings become substantially more dispersed, resulting in fewer dominated vertices. For example, the average query cost decreases from 1,182.6 (resp. 1,403.97) to 522.28 (resp. 1,027.27) for VLE (resp. VSE).

Optimizations. We further apply the following optimizations during vertex embedding training.

Optimization via L_1 Normalization. Although optimizing the structure embeddings VSE effectively reduces query cost, the average cost of label embeddings VLE remains relatively high (522.28), still exceeding the theoretical lower bound $\frac{|V(G)|}{|L(G)|} = 3{,}112/71 = 43.83$. This indicates room for further refinement. To enhance discriminability, we apply L_1 normalization to VLE embeddings:

$$o'_l(v_i) = \frac{o_l(v_i)}{\|o_l(v_i)\|_1}, \tag{8}$$

which constrains embeddings to a constant-L_1 manifold and spreads them along the anti-diagonal direction.

Empirically, as shown in Figure 6(c), L_1 normalization yields a clear anti-diagonal alignment of VLE embeddings and significantly reduces the average query cost (*e.g.,* from 522.28 to 284.6) on the Yeast dataset. The remaining gap to the theoretical lower bound $\frac{|V(G)|}{|L(G)|}$ is mainly due to label distribution skew.

Optimization of α and β. We refine the vertex embedding $o(v_i) = \alpha\, o_l(v_i) + \beta\, o_s(v_i)$ by tuning the weighting parameters α and β. Since the optimized VLE embeddings exhibit strong global separability, we treat them as the base distribution and use VSE as a fine-grained structural perturbation. Setting $\alpha \gg \beta$ (*e.g.,* $\alpha = 1000; \beta = 0.01$) preserves the favorable VLE distribution while incorporating local structural cues. As illustrated in Figures 7(a) and 7(b), increasing the α/β ratio substantially reduces dominating-region overlap and lowers the average query cost from 732.6 to 103.34.

4 *iLabel*: A Dominance-Preserving Index for Monotonic Vertex Embeddings

The monotonic vertex embeddings in Section 3 enable exact vertex-level filtering via dominance relationships, but efficiently enumerating dominating vertices remains challenging for high-dimensional embeddings on large graphs. To address this, we propose *iLabel*, a lightweight index that preserves dominance relationships while mapping multi-dimensional embeddings into a one-dimensional key space, enabling efficient range-based candidate retrieval.

4.1 Design of the *iLabel* Index

The design of *iLabel* exploits two properties of the learned monotonic vertex embeddings: (i) vertex label embeddings (VLE) naturally form semantic clusters with identical vertex labels, and (ii) the final embedding ($o(\cdot)$) is a weighted combination of label and structural components with $\alpha \gg \beta$, enabling non-overlapping clusters.

Label-based Clustering Recall that in the monotonic embedding design (Section 3.2), the vertex label embedding (VLE) $o_l(v)$ is solely determined by the vertex label. Consequently, all vertices sharing the same label are mapped to the same VLE vector, naturally partitioning data vertices into disjoint label-based clusters:

$$C = \{C_1, C_2, \ldots, C_m\}, \qquad C_i = \{v_j \mid L(v_j) = l_i\}, \tag{9}$$

where each cluster C_i consists of vertices in G with label l_i.

For each cluster C_i, we define its center as the corresponding label embedding: $c_i = o_l(v_j), \forall v_j \in C_i$. Within a cluster, vertices differ only in their structural embeddings $o_s(v_j)$, which capture local neighborhood variations around the same semantic label.

During query processing, label-based clustering restricts candidate retrieval to clusters with labels compatible with the query vertex, thereby avoiding unnecessary access to irrelevant vertices.

One-dimensional Key Mapping. Recall that each vertex embedding is constructed as $o(v) = \alpha\, o_l(v) + \beta\, o_s(v)$, where α and β are non-negative weights with $\alpha \gg \beta$. This design ensures that inter-cluster separation is dominated by label embeddings, while intra-cluster ordering is governed by structural embeddings.

iLabel maps each vertex embedding to a one-dimensional key:

$$key(v) = \alpha \|o_l(v)\|_2 + \beta \|o_s(v)\|_2, \tag{10}$$

where the first term defines a label-dependent base offset in the key space, and the second term introduces a fine-grained ordering among vertices sharing the same label. This mapping projects high-dimensional monotonic embeddings into a one-dimensional space while preserving their relative dominance order.

We next show that this key mapping is sound for dominance-based pruning, *i.e.,* dominance in the embedding space implies a consistent ordering in the key space.

Lemma 3. ***(Dominance Preservation under One-dimensional Key Mapping).*** *Let v_i and v_j be two data vertices with monotonic vertex embeddings $o(v_i)$ and $o(v_j)$. If $o(v_i) \preceq o(v_j)$ holds, then their corresponding keys satisfy $key(v_i) \leq key(v_j)$.* □

Strict Separation of Label Clusters in the Key Space. While Lemma 3 guarantees that dominance relations are preserved under the one-dimensional key mapping, efficient index traversal further benefits from a stronger property: key ranges of different label clusters should be strictly non-overlapping. Intuitively, choosing α sufficiently larger than β (*i.e.,* $\alpha \gg \beta$) ensures that label semantics dominate inter-cluster separation, while preserving structural discrimination within each cluster. We formalize this intuition by giving a sufficient condition under which label-cluster key ranges in the *iLabel* index are guaranteed to be strictly non-overlapping.

Lemma 4. ***(Non-overlapping Condition for Label Clusters).*** *Let $M = \max_{v_i \in V(G)} \|o_s(v_i)\|_2$ be the maximum L_2 norm of vertex structure embeddings (VSEs) over all data vertices, and let $\Delta_{\min} = \min_i(\|o_l(l_{i+1})\|_2 - \|o_l(l_i)\|_2)$ be the minimum difference between*

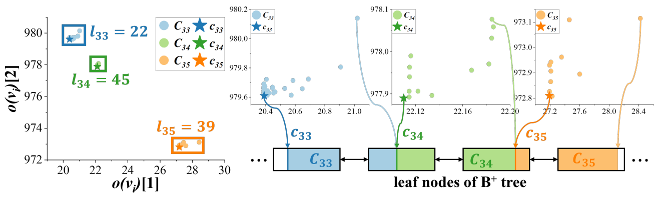
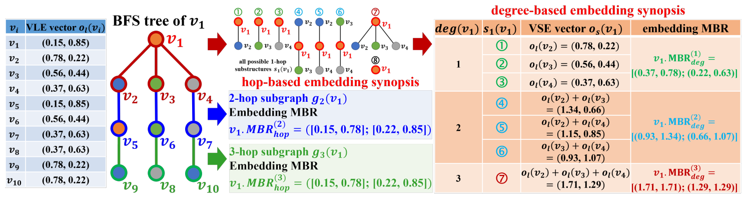


Figure 8: An illustration of our *iLabel* index design.


BFS tree of v_1
all possible 1-hop substructures $s_1(v_1)$
hop-based embedding synopsis
2-hop subgraph $g_2(v_1)$
Embedding MBR
$v_1.MBR^{(2)}_{hop}$ = ([0.15, 0.78]; [0.22, 0.85])
3-hop subgraph $g_3(v_1)$
Embedding MBR
$v_1.MBR^{(3)}_{hop}$ = ([0.15, 0.78]; [0.22, 0.85])
degree-based embedding synopsis

v_i	VLE vector $o_l(v_i)$
v_1	(0.15, 0.85)
v_2	(0.78, 0.22)
v_3	(0.56, 0.44)
v_4	(0.37, 0.63)
v_5	(0.15, 0.85)
v_6	(0.56, 0.44)
v_7	(0.37, 0.63)
v_8	(0.37, 0.63)
v_9	(0.78, 0.22)
v_{10}	(0.78, 0.22)

$deg(v_1)$	$s_1(v_1)$	VSE vector $o_s(v_1)$	embedding MBR
1	①	$o_l(v_2) = (0.78, 0.22)$	$v_1.MBR^{(1)}_{deg}$ = [(0.37, 0.78); (0.22, 0.63)]
	②	$o_l(v_3) = (0.56, 0.44)$	
	③	$o_l(v_4) = (0.37, 0.63)$	
2	④	$o_l(v_2) + o_l(v_3) = (1.34, 0.66)$	$v_1.MBR^{(2)}_{deg}$ = [(0.93, 1.34); (0.66, 1.07)]
	⑤	$o_l(v_2) + o_l(v_4) = (1.15, 0.85)$	
	⑥	$o_l(v_3) + o_l(v_4) = (0.93, 1.07)$	
3	⑦	$o_l(v_2) + o_l(v_3) + o_l(v_4) = (1.71, 1.29)$	$v_1.MBR^{(3)}_{deg}$ = [(1.71, 1.71); (1.29, 1.29)]

Figure 9: An illustration of our auxiliary synopsis design.

the L_2 norms of vertex label embeddings (VLEs) of two adjacent labels, where labels are ordered by increasing $\|o_l(l)\|_2$. If the following holds:

$$\frac{\alpha}{\beta} > \frac{M}{\Delta_{\min}}, \tag{11}$$

then the key ranges of any two distinct label clusters in the iLabel index are strictly non-overlapping. □

In practice, the value of M can be obtained by a single linear scan during offline index construction, as $\|o_s(v_i)\|_2$ is already computed for key generation. Similarly, $\Delta_{\min}$ can be computed after sorting label embeddings by their L_2 norms. Thus, the condition in Eq. 14 can be efficiently verified and enforced in the offline phase.

Example 3. *Figure 8 illustrates how iLabel organizes monotonic vertex embeddings via label-based clustering and one-dimensional key mapping. In this example, vertices have three labels, forming clusters C_{33}, C_{34}, and C_{35} corresponding to labels l_{33}, l_{34}, and l_{35}, respectively. Each cluster is anchored by its label embedding, serving as the cluster center, and vertices within a cluster share the same label embedding while differing in structural embeddings.*

Following Eq. 10, each vertex is mapped to a one-dimensional key $key(v_i) = \alpha\|o_l(v_i)\|_2 + \beta\|o_s(v_i)\|_2$ with $\alpha \gg \beta$. The first term determines the cluster offset, while the second induces a fine-grained ordering within the cluster. Consequently, key ranges of different clusters are strictly separated (e.g., $\max C_{33} < \min C_{34}$)*, enabling range queries to be performed independently within each cluster. Meanwhile, variations in the structural component $\beta\|o_s(v_i)\|_2$ order vertices within the same cluster, enabling efficient enumeration of candidates that satisfy dominance-based filtering.* □

4.2 Index Construction

Based on the *iLabel* key mapping in Section 4.1, this subsection describes how the index is constructed and organized, and how auxiliary information is incorporated to improve pruning efficiency during traversal. The index structure is solely responsible for organizing vertex keys and supporting efficient range search, while all auxiliary synopses are precomputed offline and used only as lightweight pruning aids during online query processing.

Index Construction. The construction of the *iLabel* index is performed entirely offline. After training the embedding model Emb(·) using the cost-driven objective in Section 3.2, we generate the monotonic vertex embedding $o(v_i)$ for each data vertex $v_i \in V(G)$ and map it to a one-dimensional key using Eq. 10. A B⁺-tree index $\mathcal{I}$ is then built over these keys to support efficient range-based access.

The *iLabel* index follows a standard B⁺-tree organization. Each leaf node stores data vertices together with their key values and vertex identifiers, while non-leaf nodes maintain routing entries that record the key ranges of their child subtrees. Non-leaf nodes store only key ranges, with no aggregated embedding or structural information, enabling traversal solely via key comparisons and keeping the index lightweight and scalable.

Auxiliary Synopses for Pruning. While key-based range search effectively narrows candidate retrieval, dominance-based filtering alone may still yield false positives. Thus, we precompute two types of auxiliary embedding synopses for each data vertex; these synopses are lightweight geometric summaries that enable early pruning during index traversal without introducing false dismissals.

Hop-based Embedding Synopsis. We construct a *Minimum Bounding Rectangle* (MBR)–based embedding synopsis $v_i.MBR^{(t)}_{hop}$ for each vertex v_i and hop level t ($t \geq 1$), to capture higher-order neighborhood semantics. For the t-hop subgraph $g_t(v_i)$ centered at v_i, we compute per-dimension lower and upper bounds over the vertex label embeddings of all neighbor vertices of v_i within $g_t(v_i)$:

$$\begin{aligned} v_i.MBR^{(t)}_{hop}[2k] &= \min_{v_j \in V(g_t(v_i))} o_l(v_j)[k], \\ v_i.MBR^{(t)}_{hop}[2k+1] &= \max_{v_j \in V(g_t(v_i))} o_l(v_j)[k], \end{aligned} \tag{12}$$

for $k \in [0, d)$. The resulting $2d$-dimensional vector bounds the t-hop neighborhood embeddings, enabling pruning based on multi-hop semantic inconsistency without explicit neighborhood traversal.

Degree-based Embedding Synopsis. We also construct a degree-based embedding synopsis $v_i.MBR^{(\delta)}_{deg}$ to summarize degree-δ neighbor combinations within the 1-hop neighborhood of a vertex v_i. Specifically, given the 1-hop neighbors $\mathcal{N}_1(v_i)$ of the vertex v_i, we maintain sorted lists of their label embedding values, denoted by $v_i.vle\_list_k$, for each embedding dimension $k \in [0, d)$. For a specified substructure degree δ, the synopsis records lower and upper bounds on the aggregated structural embeddings of all possible δ-neighbor combinations:

$$\begin{aligned} v_i.MBR^{(\delta)}_{deg}[2k] &= \sum_{r=1}^{\delta} v_i.vle\_list_k[r], \\ v_i.MBR^{(\delta)}_{deg}[2k+1] &= \sum_{r=\deg(v_i)-\delta+1}^{\deg(v_i)} v_i.vle\_list_k[r], \end{aligned} \tag{13}$$

for $k \in [0, d)$. This bounds the structural embeddings of all degree-δ 1-hop substructures, avoiding explicit enumeration of substructures whose number grows exponentially with vertex degree.

Example 4. *Figure 9 illustrates the construction of the degree-based embedding synopsis for a vertex v_1 with three 1-hop neighbors $\{v_2, v_3, v_4\}$, where different neighbor combinations induce different 1-hop substructures. For degree $\delta = 1$, the three possible substructures yield VSEs* (0.78, 0.22), (0.56, 0.44)*, and* (0.37, 0.63)*, resulting in $v_1.MBR^{(1)}_{deg}$ = ([0.37, 0.78]; [0.22, 0.63]). For $\delta = 2$, the three neighbor pairs produce VSEs* (1.34, 0.66), (1.15, 0.85)*, and* (0.93, 1.07)*, giving $v_1.MBR^{(2)}_{deg}$ = ([0.93, 1.34]; [0.66, 1.07]). When $\delta = 3$, all neighbors are combined, yielding a single VSE* (1.71, 1.29) *and $v_1.MBR^{(3)}_{deg}$ =* ([1.71, 1.71]; [1.29, 1.29]).

These synopses compactly bound the structural embeddings of all degree-δ ($\in [1, 3]$) 1-hop substructures of v_1. By precomputing them offline, LIVE avoids explicit substructure enumeration while enabling effective degree-aware pruning in online matching. □

Complexity Analysis. We provide the time and space complexity of constructing the *iLabel* index together with its auxiliary synopses as follows. Due to space limitations, please refer to Appendices B.1 and B.2 in [5] for detailed derivations.

Time. It needs $O(d \cdot |E(G)| + t \cdot (|V(G)| + |E(G)|) + \sum_{v_i \in V(G)} d \cdot deg(v_i) \log deg(v_i) + |V(G)| \log |V(G)|)$ time to build the *iLabel* index and its auxiliary information, which is near-linear in the graph size for sparse graphs and small hop number t.

Space. It needs $O(|V(G)| \cdot d \cdot t + d \cdot |E(G)|)$ space to build the *iLabel* index and the synopses, which scales linearly with graph size and embedding dimension, making it suitable for large graphs.

5 Subgraph Matching with Monotonic Vertex Embedding

This section presents the online subgraph matching algorithm built on the learned monotonic vertex embeddings and the *iLabel* index. Given a query graph, the online phase efficiently retrieves candidate vertices for each query vertex without false dismissals and assembles valid matches through refinement.

We introduce low-cost pruning strategies for index traversal and candidate retrieval in Section 5.1, and then present the integrated online subgraph matching algorithm in Section 5.2.

5.1 Pruning Strategies

During online query processing, candidates for each query vertex are retrieved by traversing the *iLabel* index and scanning data vertices within a query-specific key range. To efficiently eliminate invalid candidates without sacrificing correctness, we apply the following pruning strategies to progressively refine the candidate set. These strategies are derived from necessary conditions under monotonic embeddings and therefore introduce no false dismissals.

Key Lower-Bound Pruning (Strategy 1). For a query vertex q_i, we first derive a query-specific lower bound $key(q_i)$ from the B$^+$-tree in the *iLabel* index. Due to the consistency between embedding dominance and key ordering, any data vertex with a key value smaller than $key(q_i)$ cannot be dominated by $o(q_i)$ and therefore cannot be a valid candidate. Such vertices, along with their corresponding index entries, can be safely pruned.

Lemma 5. ***(Key Lower-Bound Pruning).*** *Given a query vertex $q_i \in q$ with $key(q_i)$, for any data vertex $v_i \in G$ with $key(v_i)$ or index entry $N_i \in \mathcal{I}$ with key range $[N_i.min, N_i.max)$, if $key(v_i) < key(q_i)$ or $N_i.max < key(q_i)$ holds, then v_i or the entire subtree rooted at N_i can be safely pruned.* □

Key Upper-Bound Pruning (Strategy 2). After determining the lower bound of the query range, we further restrict candidate retrieval to a label-consistent key interval. Since key ranges of different label clusters are strictly disjoint, any data vertex with a key no smaller than the key of the next label (*i.e.*, $key(L(q_i) + 1)$) belongs to a different label cluster and cannot match the query vertex. Such vertices and index entries can therefore be safely pruned.

Lemma 6. ***(Key Upper-Bound Pruning).*** *Given a query vertex $q_i \in q$ with label $L(q_i)$, for any data vertex $v_i \in G$ with key $key(v_i)$ or index entry $N_i \in \mathcal{I}$ with key range $[N_i.min, N_i.max)$, if $key(v_i) \geq key(L(q_i)+1)$ or $N_i.min \geq key(L(q_i)+1)$ holds, then v_i or the entire subtree rooted at N_i can be safely pruned.* □

Embedding Dominance Pruning (Strategy 3). Within the query-specific key range, we scan data vertices and apply a dominance check against the query embedding. Any vertex whose embedding are not dominated by the query embedding cannot satisfy 1-hop containment and is safely pruned.

Lemma 7. ***(Embedding Dominance Pruning).*** *Given a query vertex q_i and a data vertex v_i, if their monotonic vertex embeddings satisfy $o(q_i) \npreceq o(v_i)$* (i.e., *$o(v_i)$ is not dominated by $o(q_i)$), then v_i cannot be a valid candidate for q_i and can be safely pruned.* □

Hop-based Synopsis Pruning (Strategy 4). Vertex-level dominance alone does not capture higher-order structural feasibility. To further prune false positives at low cost, we exploit hop-based embedding synopses that bound multi-hop neighborhoods in the embedding space. During candidate evaluation, we compare query and data synopses in a coarse-to-fine manner, starting from larger hop distances and terminating early upon detecting a violation.

Lemma 8. ***(Hop-based Synopsis Pruning).*** *Given a query vertex q_i and a data vertex v_i, let $q_i.MBR_{hop}^{(t)}$ and $v_i.MBR_{hop}^{(t)}$ denote their hop-based embedding synopses at hop distance t. If there exists $t \in \{2, \ldots, k\}$ such that $q_i.MBR_{hop}^{(t)} \nsubseteq v_i.MBR_{hop}^{(t)}$, then v_i cannot be a valid candidate for q_i and can be safely pruned.* □

Degree-based Synopsis Pruning (Strategy 5). For high-degree data vertices, explicitly enumerating 1-hop substructures is prohibitive. Instead, we leverage degree-based embedding synopses (Section 4.2) that bound the structural embeddings of all 1-hop substructures of a given degree. A necessary condition for matching a query vertex to a data vertex is that the query's structural embedding lies within the corresponding degree-based bounding region; otherwise, the data vertex can be safely pruned.

Lemma 9. ***(Degree-based Synopsis Pruning).*** *Given a query vertex q_i and a data vertex v_i, let $q_i.vse$ denote the vertex structure embedding of q_i. If $q_i.vse \notin v_i.MBR_{deg}^{(deg(q_i))}$, then v_i cannot be a valid candidate for matching q_i and can be safely pruned.* □

5.2 Online Subgraph Matching Algorithm

Algorithm 2 presents the complete online subgraph matching procedure based on monotonic vertex embeddings and the *iLabel* index $\mathcal{I}$. It consists of three stages: candidate retrieval with index traversal and pruning (lines 1–18), matching order determination (line 19), and candidate refinement (line 20).

Candidate Retrieval (Stage 1). For each query vertex $q_i \in V(q)$, the algorithm first computes its monotonic embedding $o(q_i)$ using the trained embedding model Emb$(\cdot)$ and computes the hop-based synopsis $q_i.MBR_{hop}$ with the embeddings of its hop-based neighbors (lines 1–3). To retrieve candidate data vertices, a key-range search is performed on the *iLabel* index $\mathcal{I}$. In particular, starting from the root, the algorithm descends to the first leaf node whose key range intersects the query interval $[key(q_i), key(L(q_i) + 1))$, leveraging key lower- and upper-bound pruning to skip irrelevant subtrees (lines 4–9; Lemmas 5 and 6).

When reaching the leaf level, it performs a sequential scan over leaf nodes within the query range using the linked-leaf structure

Algorithm 2: Exact Subgraph Matching with Learnable Monotonic Vertex Embedding

```
   Input: a query graph $q$, a data graph $G$, a trained embedding model Emb$(\cdot)$,
          and an iLabel index $\mathcal{I}$
   Output: a set $\mathcal{S}$ of matching subgraphs in $G$
   // Candidate retrieval for each query vertex
1  for each query vertex $q_i \in V(q)$ do
2      generate query vertex embedding $o(q_i)$ using Emb$(\cdot)$;
3      compute hop-based synopsis $q_i.MBR_{hop}$ of $q_i$;
       // Locate the first relevant leaf node
4      $N \leftarrow \mathcal{I}.root$;
5      while $N$ is not a leaf node do
6          for each entry $N_j \in N$ do
7              if $[N_j.min, N_j.max) \cap [key(q_i), key(L(q_i)+1)) \neq \emptyset$
                then
                   // Lemmas 5 and 6
8                  $N \leftarrow N_j$;
9                  break;
       // Sequential leaf scan within the query range
10     while $N$ is not null do
11         for each data vertex $v_i \in N$ do
12             if $key(v_i) \geq key(L(q_i)+1)$ then
                   // Lemma 6
13                 break;
14             if $o(q_i) \preceq o(v_i)$ then
                   // Lemma 7
15                 if $q_i.MBR_{hop}^{(t)} \subseteq v_i.MBR_{hop}^{(t)}$ for all $t \in [2,k]$ then
                       // Lemma 8
16                     if $q_i.vse \in v_i.MBR_{deg}^{(deg(q_i))}$ then
                           // Lemma 9
17                         $q_i.cand\_set.add(v_i)$;
18         $N \leftarrow N.next$;
   // Matching order generation
19 generate an ordered list $Q$ of query vertices $q_i \in q$ based on candidate set
     sizes $|q_i.cand\_set|$;
   // Candidate refinement
20 invoke a back-tracking search procedure to enumerate valid matches in $G$
     and add them to $\mathcal{S}$;
21 return $\mathcal{S}$;
```

and terminates once the upper bound is exceeded (lines 10–13). For each encountered data vertex, a cascade of pruning strategies is applied, including embedding dominance pruning (line 14; Lemma 7), hop-based synopsis pruning (line 15; Lemma 8), and degree-based synopsis pruning (lines 16–17; Lemma 9). Vertices that pass all pruning checks are added to the candidate set $q_i.cand\_set$ (line 17).

Matching Order Generation (stage 2). After constructing candidate sets for all query vertices, the algorithm determines a matching order Q to guide the refinement phase (line 19). To minimize intermediate join results, it selects the query vertex from q with the smallest candidate set as the starting point and then iteratively appends adjacent query vertices with the smallest candidate sets until all vertices are ordered.

Refinement (stage 3). Then, a backtracking-based refinement procedure [27, 45, 61] is applied to enumerate valid matches using the candidate sets and the matching order (line 20). During refinement, partial mappings are incrementally extended while enforcing injective mapping constraints and edge-consistency conditions between the query graph q and the data graph G; and all complete mappings that satisfy the query are added into the result set $\mathcal{S}$. Finally, the algorithm returns $\mathcal{S}$ as the final output (line 21).

Table 2: Parameter settings.

Parameters	Values
the dimension, d, of the vertex embedding vector	**2**, 3, 4, 5
the weight ratio, α/β	1K, 10K, **100K**, 1M
the number, t, of synopsis hops	1, **2**, 3, 4
the size, $\|V(q)\|$, of the query graph q	5, 6, **8**, 10, 12
the average degree, $avg\_deg(q)$, of the query graph q	2, **3**, 4
the number, $\|\sum\|$, of distinct labels	5, 10, **15**, 20, 25
the average degree, $avg\_deg(G)$, of the data graph G	3, 4, **5**, 6, 7
the size, $\|V(G)\|$, of the data graph G	10K, 30K, **50K**, 80K, 100K, 500K, 1M, 5M, 10M

Table 3: Statistics of real-world graph data sets.

Data Sets	$\|V(G)\|$	$\|E(G)\|$	$\|\sum\|$	$avg\_deg(G)$
Yeast (ye)	3,112	12,519	71	8.0
HPRD (hp)	9,460	34,998	307	7.4
DBLP (db)	317,080	1,049,866	15	6.6
Youtube (yt)	1,134,890	2,987,624	25	5.3
US Patents (up)	3,774,768	16,518,947	20	8.8

Complexity Analysis. The overall time complexity of Algorithm 2 is $O(\sum_{q_i \in V(q)}(deg(q_i) \cdot d + |E(q)| + k \cdot |V(q)| \cdot d + h + R_i \cdot k \cdot d) + |V(q)|^2 + \prod_{q_i \in V(q)} |q_i.cand\_set|)$, where h is the height of the *iLabel* index. Please refer to Appendix B.3 in [5] for more details.

6 Experimental Evaluation

To evaluate the effectiveness and efficiency of LIVE, we conducted extensive experiments on synthetic/real-world graph datasets.

6.1 Experiment Settings

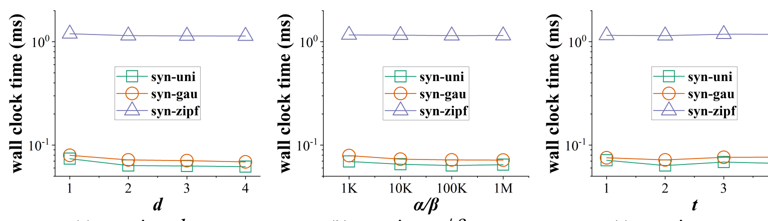
All experiments were performed on an Ubuntu server equipped with an Intel Core i9-12900K CPU, 128GB memory, and an NVIDIA GeForce RTX 4090 GPU. Offline embedding optimization and model training were implemented in PyTorch, while the online exact subgraph matching components, including index traversal, pruning, and refinement, were implemented in C++. We used the Adam optimizer with a learning rate of $\eta = 0.01$ for embedding training. To balance GPU memory usage and training cost, we fixed the number of sampled vertex pairs to $K = 4{,}096$ and trained for 1,000 epochs across all datasets. The source code and all tested datasets are available at https://anonymous.4open.science/r/LIVE-4C37.

Baseline Methods. We compared LIVE with eleven representative exact subgraph matching methods, including both classical rule-based approaches and recent learning-based solutions: GraphQL (GQL) [24], QuickSI (QSI) [45], RI [11], CFLMatch (CFL) [10], VF2++ (VF) [26], DP-iso (DP) [22], CECI [9], Hybrid [47] (a hybrid of GQL, RI, and QSI), RapidMatch (RM) [48], GNN-PE [60], and BSX [37]. These baselines collectively represent the state of the art in exact subgraph matching with different indexing, pruning, and learning strategies, providing a comprehensive comparison against LIVE.

Graph Datasets. We evaluated LIVE on both synthetic and real-world graph datasets, following widely adopted experimental settings as in prior works [47, 48, 60, 61].

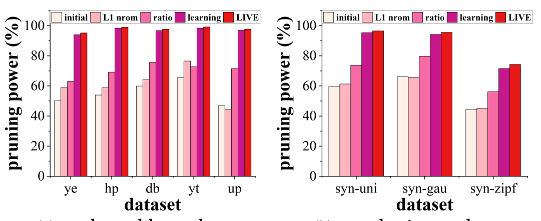
Synthetic Graphs. Following [37, 47, 60], we generated synthetic graphs using the Newman–Watts–Strogatz model [54] via the NetworkX package [21]. For each vertex v_i, its label $L(v_i)$ was randomly assigned from the range $[1, |\Sigma|]$ following one of three distributions: Uniform, Gaussian, or Zipf. Accordingly, we constructed three types of synthetic graphs, denoted as *syn-uni*, *syn-gau*, and *syn-zipf*. Detailed parameter settings are summarized in Table 2.

Real-world Graphs. We further evaluated LIVE on five real-world graph datasets that have been widely used in prior subgraph matching studies [9, 10, 22–24, 43, 45, 47, 49, 64]. The statistics of these datasets are reported in Table 3.



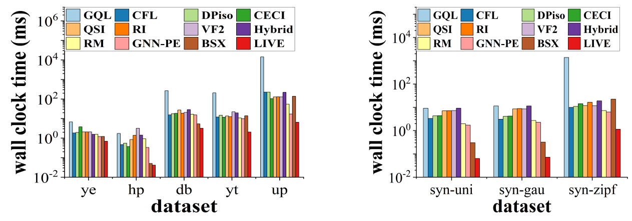
(a) varying d (b) varying α/β (c) varying t

Figure 10: LIVE efficiency w.r.t different parameters d, α/β, and t.



(a) real-world graphs (b) synthetic graphs

Figure 11: LIVE pruning power.



(a) real-world graphs (b) synthetic graphs

Figure 12: LIVE efficiency on synthetic/real-world graphs.

Query Graphs. For each data graph G, we generated 100 connected query graphs following standard practices [6, 9, 10, 22, 23, 29, 43, 49]. Each query graph q was obtained by performing a random walk on G until $|V(q)|$ vertices were collected. If the induced subgraph exceeded the target average degree $avg\_deg(q)$, edges were randomly removed to match the desired density; otherwise, the sampling process was restarted. Detailed query parameters are listed in Table 2.

Evaluation Metrics. Following [37, 47, 48, 60], we evaluated both efficiency and effectiveness of LIVE and all baselines. Efficiency was measured by the wall-clock query processing time, including both filtering and refinement phases. Effectiveness was evaluated in terms of pruning power, defined as the percentage of data vertices eliminated during candidate retrieval by the proposed pruning strategies (Section 5.1).

Unless otherwise stated, all reported results were averaged over 100 query graphs. In addition, we reported the offline pre-computation costs of LIVE, including embedding model training time, *iLabel* index construction time, and *iLabel* index storage overhead, to demonstrate the practicality of the proposed framework.

Table 2 summarizes the default parameter settings used in our experiments, where default values are highlighted in bold. In subsequent experiments, we vary one parameter at a time while keeping all others fixed at their default values.

6.2 Parameter Tuning

In this subsection, we studied the impact of key parameters in LIVE on query efficiency using synthetic datasets, varying one parameter at a time while holding the others at their default values.

Impact of Vertex Embedding Dimension d. Figure 10(a) reports the query processing time of LIVE as the embedding dimension d varied from 1 to 4. As shown there, increasing d from 1 to 2 significantly reduces query time due to stronger pruning from more expressive embeddings , while further increases yield marginal benefits as pruning saturated and per-comparison cost grew linearly. This indicates that low-dimensional monotonic vertex embeddings (*e.g.,* $d = 2$) can support efficient subgraph matching on widely tested datasets. Nevertheless, LIVE maintains consistently low query latency across different values of d (*e.g.,* below 0.07*ms* on *syn-uni* and *syn-gau*, and below 1.19*ms* on *syn-zipf*), demonstrating robustness to the embedding dimensionality.

Impact of Weight Ratio α/β. Figure 10(b) evaluated the impact of the weight ratio α/β on query efficiency, where α and β control the relative contributions of vertex label embeddings (VLE) and vertex structure embeddings (VSE), respectively. As α/β increases from 10^3 to 10^5, query time decreases and then stabilizes when α/β exceeds 10^5. This behavior aligns with the design of *iLabel*: (i) a larger α/β enhances separation among label-based clusters in the key space, resulting in tighter query ranges and fewer vertices examined during index traversal; and (ii) an effective α/β can be achieved with a relatively small value (e.g., 10^5). Overall, LIVE maintains low query latency across a wide range of α/β values (ranging from 0.06*ms* to 1.16*ms*).

Impact of Synopsis Hop Parameter t. Figure 10(c) examined the effect of the hop parameter t in the hop-based embedding synopsis, which controls the amount of multi-hop structural information used for pruning. As shown in the figure, query time first decreases as t increases from 1 to 2, and then increases for larger values of t. This behavior is expected: while a larger t enables stronger pruning by incorporating richer multi-hop information, the overhead of comparing higher-dimensional synopses eventually outweighs the additional pruning benefit. The resulting U-shaped trend reflects the trade-off between pruning effectiveness and comparison cost. Nevertheless, LIVE maintains low query latency across all settings, ranging from 0.06*ms* to 1.18*ms*.

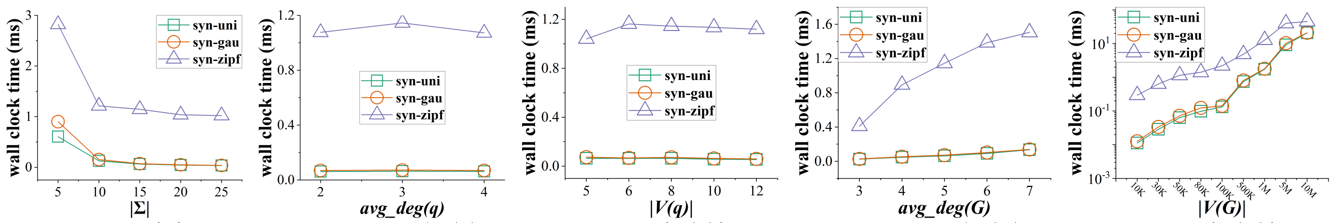
Default Parameter Selection. Based on the above observations, we set $d = 2$, $\alpha/\beta = 100\text{K}$, and $t = 2$ as the default parameters in the remaining experiments. These values strike a good balance between pruning effectiveness and computational overhead, and provide stable performance across different datasets and query workloads.

6.3 Evaluation of LIVE Effectiveness

In this subsection, we evaluated the effectiveness of LIVE in terms of pruning power, measured as the percentage of data vertices safely eliminated during candidate retrieval.

Pruning Power of LIVE on Real-world and Synthetic Graphs. Figure 11 evaluated the pruning power of LIVE under different embedding configurations and optimizations on both real-world and synthetic graphs, with all parameters set to their default values.

(i) Initial embeddings (*initial*). Vertex embeddings are randomly initialized. While monotonicity guarantees correctness, pruning power is limited (*e.g.,* only up to 66.32% of data vertices are eliminated). (ii) Lightweight optimizations (*L1 norm* and *ratio*). These apply non-learning-based optimizations, including L_1 normalization of vertex label embeddings and tuning the weight ratio α/β. Both reshape the embedding distribution: L_1 normalization reduces



(a) varying $|\Sigma|$ (b) varying $avg\_deg(q)$ (c) varying $|V(q)|$ (d) varying $avg\_deg(G)$ (e) varying $|V(G)|$

Figure 13: LIVE efficiency evaluation on synthetic graphs.

dominance region size, while adjusting α/β strengthens label-based separation in the *iLabel* key space, yielding consistent but moderate gains (*e.g.,* up to 76.49% and 79.73% pruning). (iii) Embedding learning (*learning*). Optimizing embeddings with the proposed anti-dominance loss significantly increases dispersion in the embedding space, resulting in much stronger pruning (*e.g.,* up to 98.43%). (iv) Full LIVE pruning (*LIVE*). Combining embedding learning with all pruning strategies, *i.e.,* key-based, dominance-based, hop-based, and degree-based pruning, further eliminates false positives and achieves high pruning power, up to 99.18%, removing over 99% of data vertices before refinement.

These results confirm the effectiveness of LIVE's embedding design and pruning strategies. More importantly, they highlight a clear separation between correctness guarantees and pruning effectiveness. While monotonicity alone ensures no false dismissals (even with randomly initialized embeddings), strong pruning power is achieved only when vertex embeddings are explicitly optimized to reduce dominance. This empirical evidence shows that dominance correctness by itself does not imply efficient filtering, and validates the core design of LIVE: decoupling correctness guarantees from pruning-oriented embedding optimization.

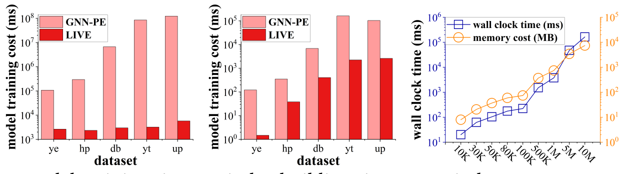
6.4 Evaluation of LIVE Efficiency

In this subsection, we evaluated the efficiency of LIVE in terms of wall-clock time, and compared it with state-of-the-art baseline methods on both real-world and synthetic graphs. Unless otherwise stated, all parameters are set to their default values.

Efficiency of LIVE on Synthetic and Real-world Graphs. Figure 12 compared the time cost of LIVE with eleven baselines on both synthetic and real-world datasets. As shown, LIVE consistently outperforms all baselines across all datasets, achieving up to an order-of-magnitude speedup over the strongest competitors on large-scale real-world graphs such as *yt* and *up*. Notably, LIVE answers exact subgraph matching queries on the largest *up*, which contains $3.77M$ vertices, in only $6.53ms$. These results demonstrate the high efficiency of LIVE and its strong scalability to large graphs.

To further evaluate the efficiency of LIVE, we varied key parameters (e.g., $|\Sigma|$, $avg\_deg(q)$, $|V(q)|$, $avg\_deg(G)$, and $|V(G)|$) on synthetic graphs in subsequent tests. To better illustrate performance trends, baseline results are omitted below.

Efficiency of LIVE w.r.t. # of Distinct Vertex Labels, $|\Sigma|$. Figure 13(a) shows the query time of LIVE as the number of distinct vertex labels $|\Sigma|$ increases from 5 to 25. As $|\Sigma|$ increases, label-based clustering in the *iLabel* index becomes more effective, leading to smaller query ranges and fewer candidate vertices, and thus lower query time. Although different label distributions introduce some variation in pruning behavior, LIVE consistently maintains low query latency across all settings, ranging from $0.03ms$ to $2.82ms$.



(a) model training time (b) index building time (c) index storage cost

Figure 14: The LIVE offline pre-computation cost.

Efficiency of LIVE w.r.t. Average Degree, $avg\_deg(q)$, of the Query Graph q. Figure 13(b) evaluated the impact of the query graph's average degree on efficiency. As $avg\_deg(q)$ increases, query time first rises and then declines. This is because although higher query density enables stronger pruning due to tighter matching requirements, it introduces additional overhead in computing hop-based synopses and enforcing structural constraints. Nevertheless, query time remains low across all settings ($0.06ms \sim 1.15ms$).

Efficiency of LIVE w.r.t. Query Graph Size $|V(q)|$. Figure 13(c) reports query time as the query graph size increased from 5 to 12 vertices. Intuitively, larger queries involve more vertices and can increase index traversal and refinement costs, but they also yield richer hop-based synopses that strengthen pruning and reduce candidate sets. As a result, LIVE exhibits decreasing query time as $|V(q)|$ grows, indicating that stronger structural constraints effectively offset the additional processing overhead. Across all query sizes, query time remains low, ranging from $0.05ms$ to $1.16ms$.

Efficiency of LIVE w.r.t. Average Degree, $avg\_deg(G)$, of the Data Graph G. Figure 13(d) evaluated LIVE under varying data graph densities, with $avg\_deg(G)$ from 3 to 7. As shown in the figure, query time increases only moderately as graph density grows, even on graphs with skewed label distributions, *e.g.,* from $0.41ms$ to $1.51ms$ on *syn-zipf*. This trend arises because higher density weakens pruning effectiveness and enlarges candidate sets. LIVE mitigates this effect through degree-based synopsis pruning, which restricts structural checks to substructures whose degrees match those of query vertices. Overall, LIVE is efficient across all settings.

Scalability of LIVE w.r.t. Data Graph Size $|V(G)|$. Figure 13(e) evaluated the scalability of LIVE as the data graph size increased from $10K$ to $10M$ vertices. LIVE consistently achieves the lowest query time across all settings, *e.g.,* it answers exact subgraph matching queries within $45.26ms$ on graphs with $10M$ vertices. This efficiency stems from the one-dimensional *iLabel* index design combined with effective pruning strategies, which tightly bound candidate exploration even as graph size grows. This indicates that LIVE scales well for exact subgraph matching on large-scale graphs.

6.5 Offline Pre-Computation Performance

In this subsection, we evaluated the offline pre-computation cost of LIVE, including embedding model training and index construction, and compared it with the representative learning-based baseline GNN-PE. Experiments were conducted on both real-world and synthetic graphs using default parameter settings.

Offline Pre-Computation Cost on Real-world Graphs. Figures 14(a) and 14(b) compared the model training and index construction costs of LIVE and GNN-PE on real-world datasets.

As shown in Figure 14(a), GNN-PE incurs $10^4\times$ higher training cost, which increases sharply with graph size due to exhaustive enumeration of 1-hop subgraphs and their substructures for learning dominance relations. In contrast, LIVE decouples dominance correctness from the learning objective and adopts a sampling-based anti-dominance loss with a fixed number of sampled vertex pairs, resulting in low and stable training cost across datasets, with only minor overhead from embedding computation. Figure 14(b) further shows that although index construction time increases with graph size for both methods, LIVE consistently achieves 82× lower cost by building a lightweight *iLabel* index over vertex embeddings, whereas GNN-PE relies on path-based structures that substantially inflate index size and construction overhead.

These results demonstrate the superior cost performance of LIVE in both model training and index construction on large-scale graphs. More importantly, they highlight a fundamental limitation of dominance learning-based approach (*e.g.,* GNN-PE): by tightly coupling correctness guarantees with the learning objective, such methods inevitably incur prohibitively high training costs due to exhaustive subgraph–substructure enumeration. In contrast, LIVE attains the same no-false-dismissal guarantee through monotonic embedding construction and decouples correctness from embedding optimization. This structural decoupling enables lightweight, sampling-based training and avoids the offline cost bottleneck inherent in dominance learning-based frameworks.

Index Construction Time and Space Costs of LIVE w.r.t. Data Graph Size $|V(G)|$. Figure 14(c) further evaluated the time and space overhead of constructing the *iLabel* index on synthetic graphs as the data graph size $|V(G)|$ increases from $10K$ to $10M$. Across this range, the total index construction time, including vertex embedding generation, auxiliary synopsis computation, and B^+-tree construction, grows from $19.98ms$ to $163.59sec$. Correspondingly, index storage increases from $7.93MB$ to $7{,}400.51MB$. These results indicate that the offline cost of LIVE scales gracefully with graph size and remains practical for large-scale graphs.

7 Related Work

We classify existing works as follows.

Exact Subgraph Matching. Existing exact subgraph matching methods fall into two categories: (i) join-based algorithms and (ii) backtracking-search-based algorithms. Join-based methods [1, 4, 30–32, 39, 48] decompose a query graph into smaller substructures (*e.g.,* edges, paths, or triangles), enumerate matches for each substructure independently, and join these partial results to obtain final matches. Backtracking-based algorithms [9–11, 13, 14, 22, 24, 45, 49] first compute candidate sets for query vertices using filtering techniques, and then enumerate exact matches via depth-first search while enforcing injective mapping and edge-consistency constraints; recent work further applies reinforcement learning to optimize matching orders in backtracking-based frameworks [52, 59].

LIVE differs from prior work in three key aspects: (i) it scales to dense queries and large graphs via effective pruning with the *iLabel* index, avoiding the large intermediates of join-based methods [1, 4, 30–32, 39, 48]; (ii) it relies on learned vertex embeddings rather than explicit structural comparisons, improving scalability over backtracking-based algorithms [9–11, 13, 14, 22, 24, 45, 49]; and (iii) unlike feature-vector graph search targeting small graphs [25, 46, 58], LIVE enumerates all exact matches within a single large graph.

Approximate Subgraph Matching. Another line of work studies approximate subgraph matching, which aims to efficiently retrieve subgraphs that are similar to a given query graph rather than exactly isomorphic. To improve scalability, these methods relax exact structural constraints and employ heuristic or distance-based similarity measures, such as graph edit distance [34], neighborhood similarity [55], or feature-based approximations [17, 18]. As a result, approximate matching algorithms trade accuracy for efficiency and may return false positives or miss valid exact matches.

In contrast, the problem addressed in this work requires enumerating all subgraphs that are exactly isomorphic to the query graph, with no false positives or false dismissals. Therefore, approximate subgraph matching methods do not provide the correctness guarantees required here and are orthogonal to the focus of LIVE.

Learning-based Subgraph Matching. Recent work explores learning techniques to accelerate subgraph matching by avoiding explicit graph-structure comparisons. Early learning-based approaches focus on approximate matching, using GNNs or DNNs to embed graphs or subgraphs into embedding spaces and compare them via distance functions [8, 33, 36, 38, 51, 56]. More recently, learning has been applied to *exact* subgraph matching; for example, GNN-PE [60] learns dominance-preserving embeddings for graph paths, enabling candidate filtering without false dismissals and representing the first learning-based framework with correctness guarantees.

This work differs from prior studies in two aspects. (i) LIVE provides correctness guarantees for exact subgraph isomorphism, unlike heuristic learning-based approaches [8, 33, 36, 38, 51, 56]. (ii) LIVE enforces dominance preservation by construction through monotonic vertex embedding design, decoupling correctness from learning objectives and achieving lower training cost and stronger pruning effectiveness than prior learning-based methods (*e.g.,* [60]), which rely on costly enumeration of subgraph–substructure pairs.

8 Conclusion

In this paper, we propose LIVE, a learning-based framework for exact subgraph matching on large graphs that achieves both scalability and exactness. By enforcing monotonicity in vertex embedding construction, LIVE guarantees dominance correctness by design and fundamentally decouples no-false-dismissal guarantees from embedding optimization. This decoupling enables candidate filtering without false dismissals while allowing embedding learning to directly focus on improving vertex-level pruning power. We further introduce a query cost model with a differentiable surrogate objective to guide offline embedding training, and design a lightweight one-dimensional *iLabel* index that preserves dominance relationships and supports efficient candidate retrieval without complex indexing structures. Extensive experiments on synthetic and real-world datasets validate the performance of LIVE.

References

[1] Christopher R Aberger, Andrew Lamb, Susan Tu, Andres Nötzli, Kunle Olukotun, and Christopher Ré. 2017. Emptyheaded: A relational engine for graph processing. *ACM Transactions on Database Systems* 42, 4 (2017), 1–44.

[2] Ahmed Al-Baghdadi and Xiang Lian. 2020. Topic-based community search over spatial-social networks. In *Proceedings of the International Conference on Very Large Data Bases (PVLDB).* 2104–2117.

[3] Uri Alon. 2007. Network motifs: theory and experimental approaches. *Nature Reviews Genetics* 8, 6 (2007), 450–461.

[4] Khaled Ammar, Frank McSherry, Semih Salihoglu, and Manas Joglekar. 2018. Distributed Evaluation of Subgraph Queries Using Worst-case Optimal Low-Memory Dataflows. In *Proceedings of the International Conference on Very Large Data Bases (PVLDB).* 691–704.

[5] anonymous authors. 2026. Appendix. *https://anonymous.4open.science/r/LIVE-4C37* (2026).

[6] Blair Archibald, Fraser Dunlop, Ruth Hoffmann, Ciaran McCreesh, Patrick Prosser, and James Trimble. 2019. Sequential and parallel solution-biased search for subgraph algorithms. In *Proceedings of the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR).* 20–38.

[7] László Babai. 2018. Group, graphs, algorithms: the graph isomorphism problem. In *Proceedings of the International Congress of Mathematicians: Rio de Janeiro 2018.* World Scientific, 3319–3336.

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

[9] Bibek Bhattarai, Hang Liu, and H Howie Huang. 2019. Ceci: Compact embedding cluster index for scalable subgraph matching. In *Proceedings of the International Conference on Management of Data (SIGMOD).* 1447–1462.

[10] Fei Bi, Lijun Chang, Xuemin Lin, Lu Qin, and Wenjie Zhang. 2016. Efficient subgraph matching by postponing cartesian products. In *Proceedings of the International Conference on Management of Data (SIGMOD).* 1199–1214.

[11] Vincenzo Bonnici, Rosalba Giugno, Alfredo Pulvirenti, Dennis Shasha, and Alfredo Ferro. 2013. A subgraph isomorphism algorithm and its application to biochemical data. *BMC Bioinformatics* 14, 7 (2013), 1–13.

[12] Stephan Borzsony, Donald Kossmann, and Konrad Stocker. 2001. The skyline operator. In *Proceedings of the International Conference on Data Engineering (ICDE).* 421–430.

[13] Vincenzo Carletti, Pasquale Foggia, Alessia Saggese, and Mario Vento. 2017. Challenging the time complexity of exact subgraph isomorphism for huge and dense graphs with VF3. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 40, 4 (2017), 804–818.

[14] Vincenzo Carletti, Pasquale Foggia, and Mario Vento. 2015. VF2 Plus: An improved version of VF2 for biological graphs. In *International Workshop on Graph-Based Representations in Pattern Recognition (GbRPR).* 168–177.

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

[16] Alin Deutsch, Nadime Francis, Alastair Green, Keith Hare, Bei Li, Leonid Libkin, Tobias Lindaaker, Victor Marsault, Wim Martens, Jan Michels, et al. 2022. Graph pattern matching in GQL and SQL/PGQ. In *Proceedings of the International Conference on Management of Data (SIGMOD).* 2246–2258.

[17] Boxin Du, Si Zhang, Nan Cao, and Hanghang Tong. 2017. First: Fast interactive attributed subgraph matching. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (SIGKDD).* 1447–1456.

[18] Sourav Dutta, Pratik Nayek, and Arnab Bhattacharya. 2017. Neighbor-aware search for approximate labeled graph matching using the chi-square statistics. In *Proceedings of the Web Conference (WWW).* 1281–1290.

[19] Michael R. Garey and David S. Johnson. 1983. Computers and intractability: A guide to the theory of NP-completeness. *The Journal of Symbolic Logic* 48, 2 (1983), 498–500.

[20] Martin Grohe and Pascal Schweitzer. 2020. The graph isomorphism problem. *Commun. ACM* 63, 11 (2020), 128–134.

[21] Aric Hagberg and Drew Conway. 2020. Networkx: Network analysis with python. *URL: https://networkx. github. io* (2020).

[22] 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 *Proceedings of the International Conference on Management of Data (SIGMOD).* 1429–1446.

[23] Wook-Shin Han, Jinsoo Lee, and Jeong-Hoon Lee. 2013. Turboiso: towards ultrafast and robust subgraph isomorphism search in large graph databases. In *Proceedings of the International Conference on Management of Data (SIGMOD).* 337–348.

[24] Huahai He and Ambuj K Singh. 2008. Graphs-at-a-time: query language and access methods for graph databases. In *Proceedings of the International Conference on Management of Data (SIGMOD).* 405–418.

[25] Craig A James. 2004. Daylight theory manual. *http://www. daylight. com/dayhtml/doc/theory/theory. toc. html* (2004).

[26] Alpár Jüttner and Péter Madarasi. 2018. VF2++—An improved subgraph isomorphism algorithm. *Discrete Applied Mathematics* 242 (2018), 69–81.

[27] Chathura Kankanamge, Siddhartha Sahu, Amine Mhedbhi, Jeremy Chen, and Semih Salihoglu. 2017. Graphflow: An active graph database. In *Proceedings of the International Conference on Management of Data (SIGMOD).* 1695–1698.

[28] Guy Karlebach and Ron Shamir. 2008. Modelling and analysis of gene regulatory networks. *Nature Reviews Molecular Cell Biology* 9, 10 (2008), 770–780.

[29] Foteini Katsarou, Nikos Ntarmos, and Peter Triantafillou. 2017. Subgraph querying with parallel use of query rewritings and alternative algorithms. In *Proceedings of the International Conference on Extending Database Technology (EDBT).* 25–36.

[30] Longbin Lai, Lu Qin, Xuemin Lin, and Lijun Chang. 2015. Scalable subgraph enumeration in mapreduce. In *Proceedings of the International Conference on Very Large Data Bases (PVLDB).* 974–985.

[31] Longbin Lai, Lu Qin, Xuemin Lin, Ying Zhang, Lijun Chang, and Shiyu Yang. 2016. Scalable distributed subgraph enumeration. In *Proceedings of the International Conference on Very Large Data Bases (PVLDB).* 217–228.

[32] Longbin Lai, Zhu Qing, Zhengyi Yang, Xin Jin, Zhengmin Lai, Ran Wang, Kongzhang Hao, Xuemin Lin, Lu Qin, Wenjie Zhang, et al. 2019. Distributed subgraph matching on timely dataflow. In *Proceedings of the International Conference on Very Large Data Bases (PVLDB).* 1099–1112.

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

[34] Zijian Li, Xun Jian, Xiang Lian, and Lei Chen. 2018. An efficient probabilistic approach for graph similarity search. In *Proceedings of the International Conference on Data Engineering (ICDE).* 533–544.

[35] Xiang Lian and Lei Chen. 2011. Efficient query answering in probabilistic RDF graphs. In *Proceedings of the International Conference on Management of Data (SIGMOD).* 157–168.

[36] Zhaoyu Lou, Jiaxuan You, Chengtao Wen, Arquimedes Canedo, Jure Leskovec, et al. 2020. Neural subgraph matching. *arXiv preprint arXiv:2007.03092* (2020).

[37] Yujie Lu, Zhijie Zhang, and Weiguo Zheng. 2025. B X: Subgraph Matching with Batch Backtracking Search. In *Proceedings of the International Conference on Management of Data (SIGMOD).* 1–27.

[38] Brian McFee and Gert Lanckriet. 2009. Partial order embedding with multiple kernels. In *Proceedings of the International Conference on Machine Learning (ICML).* 721–728.

[39] Amine Mhedbhi and Semih Salihoglu. 2019. Optimizing subgraph queries by combining binary and worst-case optimal joins. In *Proceedings of the International Conference on Very Large Data Bases (PVLDB).* 1692–1704.

[40] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS).* 1–12.

[41] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. 2017. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* 652–660.

[42] Miao Qiao, Hao Zhang, and Hong Cheng. 2017. Subgraph matching: on compression and computation. In *Proceedings of the International Conference on Very Large Data Bases (PVLDB).* 176–188.

[43] Xuguang Ren and Junhu Wang. 2015. Exploiting vertex relationships in speeding up subgraph isomorphism over large graphs. In *Proceedings of the International Conference on Very Large Data Bases (PVLDB).* 617–628.

[44] Siddhartha Sahu, Amine Mhedbhi, Semih Salihoglu, Jimmy Lin, and M Tamer Özsu. 2017. The ubiquity of large graphs and surprising challenges of graph processing. In *Proceedings of the International Conference on Very Large Data Bases (PVLDB).* 420–431.

[45] Haichuan Shang, Ying Zhang, Xuemin Lin, and Jeffrey Xu Yu. 2008. Taming verification hardness: an efficient algorithm for testing subgraph isomorphism. In *Proceedings of the International Conference on Very Large Data Bases (PVLDB).* 364–375.

[46] Dennis Shasha, Jason TL Wang, and Rosalba Giugno. 2002. Algorithmics and applications of tree and graph searching. In *Proceedings of the Principles of Database Systems (PODS).* 39–52.

[47] Shixuan Sun and Qiong Luo. 2020. In-memory subgraph matching: An in-depth study. In *Proceedings of the International Conference on Management of Data (SIGMOD).* 1083–1098.

[48] Shixuan Sun, Xibo Sun, Yulin Che, Qiong Luo, and Bingsheng He. 2020. Rapidmatch: A holistic approach to subgraph query processing. In *Proceedings of the International Conference on Very Large Data Bases (PVLDB).* 176–188.

[49] Zhao Sun, Hongzhi Wang, Haixun Wang, Bin Shao, and Jianzhong Li. 2012. Efficient Subgraph Matching on Billion Node Graphs. In *Proceedings of the International Conference on Very Large Data Bases (PVLDB).* 788–799.

[50] Damian Szklarczyk et al. 2015. STRING v10: protein–protein interaction networks, integrated over the tree of life. *Nucleic Acids Research* 43, D1 (2015),

D447–D452.

[51] Ivan Vendrov, Ryan Kiros, Sanja Fidler, and Raquel Urtasun. 2016. Order-embeddings of images and language. In *Proceedings of the International Conference on Learning Representations (ICLR)*. 1–12.

[52] 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 *Proceedings of the International Conference on Data Engineering (ICDE)*. 245–258.

[53] Stanley Wasserman and Katherine Faust. 1994. Social network analysis: Methods and applications. (1994).

[54] Duncan J Watts and Steven H Strogatz. 1998. Collective dynamics of 'small-world'networks. *Nature* 393, 6684 (1998), 440–442.

[55] Qi Wen, Yutong Ye, Xiang Lian, and Mingsong Chen. 2025. S3AND: Efficient Subgraph Similarity Search Under Aggregated Neighbor Difference Semantics. In *Proceedings of the International Conference on Very Large Data Bases (PVLDB)*. 3708–3720.

[56] Kun Xu, Liwei Wang, Mo Yu, Yansong Feng, Yan Song, Zhiguo Wang, and Dong Yu. 2019. Cross-lingual knowledge graph alignment via graph matching neural network. *arXiv preprint arXiv:1905.11605* (2019).

[57] Xifeng Yan, Hong Cheng, Jiawei Han, and Philip S Yu. 2008. Mining significant graph patterns by leap search. In *Proceedings of the International Conference on Management of Data (SIGMOD)*. 433–444.

[58] Xifeng Yan, Philip S Yu, and Jiawei Han. 2004. Graph indexing: a frequent structure-based approach. In *Proceedings of the International Conference on Management of Data (SIGMOD)*. 335–346.

[59] Linglin Yang, Lei Zou, and Chunshan Zhao. 2025. NeuSO: Neural Optimizer for Subgraph Queries. In *Proceedings of the International Conference on Management of Data (SIGMOD)*. 1–28.

[60] Yutong Ye, Xiang Lian, and Mingsong Chen. 2024. Efficient Exact Subgraph Matching via GNN-based Path Dominance Embedding. In *Proceedings of the International Conference on Very Large Data Bases (PVLDB)*. 1628–1641.

[61] Yutong Ye, Xiang Lian, Nan Zhang, and Mingsong Chen. 2025. Continuous Subgraph Matching via Cost-Model-based Dynamic Vertex Dominance Embeddings. In *Proceedings of the International Conference on Management of Data (SIGMOD)*. 1–27.

[62] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan Salakhutdinov, and Alexander Smola. 2017. Deep Sets. In *Advances in Neural Information Processing Systems (NeurIPS)*, Vol. 30.

[63] Nan Zhang, Yutong Ye, Xiang Lian, and Mingsong Chen. 2024. Top-L Most Influential Community Detection Over Social Networks. In *Proceedings of the International Conference on Data Engineering (ICDE)*. 5767–5779.

[64] Peixiang Zhao and Jiawei Han. 2010. On graph query optimization in large networks. In *Proceedings of the International Conference on Very Large Data Bases (PVLDB)*. 340–351.