COST-GED: COST-VALIDATED SELF-IMPROVEMENT FOR GRAPH EDIT DISTANCE

Anonymous authors
Paper under double-blind review

ABSTRACT

Graph edit distance (GED) is a fundamental measure of graph similarity, but computing it exactly is NP-hard. Although neural GED solvers improve inference efficiency, many still rely on exact GED values or optimal matchings as supervision, limiting large-scale training, especially on graphs with complex node and edge labels. We introduce COST-GED, a cost-validated self-improvement framework that requires neither exact GED values nor optimal matchings as training supervision. COST-GED exploits the fact that the edit cost of a feasible matching can be computed directly and provides an upper bound on GED. It updates pseudo-matchings using edit costs rather than model confidence: a candidate is accepted only if it strictly lowers the current cost, and the improved matching supervises an autoregressive solver. For each fixed graph pair and edit-cost function, this rule guarantees a monotonically non-increasing pseudo-matching cost. Global sampling and local perturbation-based reconstruction balance exploration with reuse of existing correspondences. Across six public small-graph GED benchmarks, COST-GED achieves performance competitive with supervised methods. On Code2-22, increasing the unlabeled training pool from 2K to 40K graph pairs improves the exact hit rate (EHR) from 84.5% to 95.8%. On the larger MolHIV-30-50 and MolHIV-50-70 benchmarks, COST-GED reduces the mean edit cost of returned feasible matchings by 16.92% and 19.26%, respectively, relative to IPFP, the best-performing traditional baseline in our comparison.

1 INTRODUCTION

Graph edit distance (GED) is a fundamental measure of graph similarity and has been widely used in molecular analysis Garcia-Hernandez et al. (2019), pattern recognition Riba et al. (2021), and cybersecurity Jia et al. (2024). GED is defined as the minimum cumulative cost of transforming one graph into another through node and edge insertions, deletions, and substitutions Justice & Hero (2006). Unlike similarity measures based solely on statistical features or graph-level embeddings, GED jointly captures graph topology and attribute information while providing interpretable edit paths Zeng et al. (2009).

Computing GED exactly requires searching for an optimal edit path over a combinatorially large space of node matchings and is NP-hard Blumenthal & Gamper (2020). Traditional exact approaches typically rely on A* search, branch-and-bound, or integer programming Blumenthal & Gamper (2020). Although these methods can obtain exact solutions for relatively small graph instances, their computational cost limits scalability to larger graph pairs. Early neural approaches, such as SimGNN Bai et al. (2019), formulate GED estimation as graph-pair similarity regression and predict a scalar similarity score. Such regression-based methods, however, do not explicitly recover node correspondences or edit paths and therefore do not directly produce feasible edit paths. To address these limitations, subsequent learning-based and hybrid methods, including MATA* Liu et al. (2023), GEDGNN Piao et al. (2023), and GEDHOT Cheng et al. (2025), explicitly incorporate matching, search, or cost-aware optimization to construct explicit matchings or edit paths, thereby improving structural interpretability.

Despite this progress, many neural GED methods that produce feasible solutions still rely on exact GED values or ground-truth node matchings as supervision Huang et al. (2025a). Obtaining such supervision typically requires expensive offline solvers, which limits large-scale training in realistic

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

settings. Recent advances in neural combinatorial optimization suggest that effective learning can be achieved without expert solutions through iterative candidate generation and self-improvement Pirnay & Grimm (2024). Applying this idea to GED, however, raises two key challenges. First, how can the quality of structured pseudo-matchings be assessed without exact labels? GED matchings are subject to one-to-one assignment constraints Bougleux et al. (2017), so high confidence in individual node correspondences does not necessarily imply a lower global edit cost. Moreover, in autoregressive GED solvers, early correspondence choices directly affect subsequent decoding decisions, and greedy inference can be sensitive to the initial matches Pellizzoni et al. (2026). Second, how can better matchings be continually discovered under a limited search budget? Restricting search to local modifications may lead to stagnation, whereas regenerating a matching from scratch may discard useful correspondences already present in the current pseudo-matching. Candidate generation must therefore balance global exploration with local exploitation.

To address these challenges, we propose **COST-GED**, a cost-validated self-improvement framework. For pseudo-matching quality assessment, the framework exploits a key computational distinction: finding an optimal matching is difficult, whereas the edit cost of a complete feasible matching can be computed directly. Because this cost provides an upper bound on GED, a lower-cost candidate yields a tighter upper bound without requiring the exact GED. COST-GED therefore accepts a candidate only when it strictly lowers the current pseudo-matching cost and uses the improved matching as training supervision. For each fixed graph pair and edit-cost function, this update rule guarantees that the maintained pseudo-matching cost is monotonically non-increasing. To discover better candidates under a limited search budget, COST-GED combines global sampling with local perturbation-based reconstruction. Global sampling explores alternative complete matchings with reduced dependence on the current pseudo-matching, while local reconstruction constructs candidates through partial replacement and feasibility repair while retaining existing correspondences. Candidates from both sources are evaluated using the same edit-cost acceptance rule.

Our contributions are threefold:

1. **We propose COST-GED, a cost-validated self-improvement framework that requires neither exact GED values nor optimal matchings as training supervision.** It updates pseudo-matchings through strict reductions in complete matching cost, feeds improved matchings back into training, and guarantees non-increasing pseudo-matching costs for each fixed graph pair and edit-cost function.
2. **We combine higher-temperature global sampling with local perturbation-based reconstruction using lower-temperature reference matchings.** This design explores alternative complete matchings while reusing existing correspondences. Experiments show that global sampling can alleviate local-search stagnation under a limited search budget.
3. **We evaluate COST-GED across six public small-graph GED benchmarks and two larger-graph benchmarks.** COST-GED achieves competitive performance against supervised methods on the small-graph benchmarks. On MolHIV-30-50 and MolHIV-50-70, it reduces the mean edit cost of returned feasible matchings by 16.92% and 19.26%, respectively, relative to IPFP, the best-performing traditional baseline in our comparison.

2 Related Work

Early learning-based GED methods often formulate graph similarity as a regression or similarity-learning problem. SimGNN Bai et al. (2019), Graph Matching Networks Li et al. (2019), and GraphSim Bai et al. (2020) learn graph-pair representations or node-level interactions to predict GED-derived similarity scores. These methods enable efficient inference, but typically rely on precomputed GED values for supervision and do not explicitly recover feasible node matchings or edit paths.

More recent methods incorporate the combinatorial structure of GED into neural prediction. GEDGNN Piao et al. (2023), MATA* Liu et al. (2023), Noah Yang & Zou (2021), GELATO Pellizzoni et al. (2026), and DiffGED Huang et al. (2025b) combine learned representations with node matching, search, autoregressive decoding, or generative sampling to construct explicit correspondences or candidate edit paths. While these approaches improve structural interpretability and solution feasibility, many still depend on precomputed GED values or node matchings during training.

Several recent approaches seek to reduce this dependence on ground-truth supervision. GEDRanker Huang et al. (2025a) learns from preference signals among candidate node matchings without ground-truth labels, while GEDGW Cheng et al. (2025) formulates GED approximation using unsupervised optimal-transport-based objectives and is further combined with the supervised GEDIOT model in the GEDHOT ensemble. GRAIL Verma et al. (2025) instead uses large language models to generate heuristic GED solvers without ground-truth supervision. In contrast, COST-GED directly evaluates complete feasible matchings using their computable edit costs and accepts pseudo-matching updates only when they strictly reduce the current cost. This provides training feedback aligned with the GED objective without requiring exact GED values or optimal matchings as training supervision.

3 PRELIMINARIES

A labeled graph is represented as $G = (V, E, \ell)$, where V and E denote the node and edge sets, and ℓ assigns labels to nodes and edges. Graph edit distance (GED) is the minimum cumulative cost of transforming one graph into another through node and edge insertions, deletions, and substitutions Justice & Hero (2006).

A feasible node matching is a mapping $\pi : V_1 \to V_2 \cup \{\epsilon\}$, where ϵ denotes a null node. Mapping a source node to ϵ represents deletion, while unmatched target nodes correspond to insertions. The matching must satisfy injectivity on non-null targets: distinct source nodes cannot map to the same target node. A matching π induces node and edge edit costs $C_V(\pi)$ and $C_E(\pi)$, with total cost $C_\pi(G_1, G_2) = C_V(\pi) + C_E(\pi)$.

We use unit costs for node and edge insertions and deletions, with substitution cost 1 for unequal labels and 0 otherwise. Deleting a node and its incident edges is charged separately, and each undirected edge is counted once. Under this cost model,

$$\mathrm{GED}(G_1, G_2) = \min_{\pi' \in \Pi} C_{\pi'}(G_1, G_2) \leq C_\pi(G_1, G_2), \qquad \forall\, \pi \in \Pi. \tag{1}$$

where Π denotes the set of feasible node matchings. The cost of any feasible matching can be computed directly from the graph structure, labels, and edit-cost function without knowing the exact GED. COST-GED uses these computable matching costs to evaluate candidate matchings and update pseudo-matchings.

4 A COST-VALIDATED SELF-IMPROVEMENT FRAMEWORK FOR GED

We propose COST-GED, a cost-validated self-improvement framework that derives training feedback from the edit costs of candidate matchings. The framework is based on two observations concerning cost evaluation and candidate generation. First, the edit cost of any complete feasible candidate matching can be computed directly. Second, the base solver, GELATO, can generate candidate node matchings autoregressively. COST-GED iteratively generates candidates, validates their edit costs, and updates pseudo-matchings without using exact GED values or optimal matchings as training supervision. Figure 1 illustrates this candidate generation, cost validation, and iterative update process.

COST-GED first uses the Hungarian algorithm Kuhn (1955) to initialize pseudo-matchings for cold start. In each subsequent iteration, the current solver performs global sampling and local reconstruction for each graph pair to generate candidate matchings. A candidate is accepted only if its edit cost is strictly lower than that of the current pseudo-matching. The updated pseudo-matchings are converted into link prediction training subproblems to update the solver parameters, providing feedback for subsequent candidate generation.

Candidate generation seeks complete matchings with lower edit costs than the current pseudo-matching. Because COST-GED constructs matchings autoregressively, early choices affect the available target nodes and alignment relationships, influencing subsequent decoding. Errors may therefore accumulate when labels repeat, graph structures are similar, or the model is insufficiently trained. To address this issue, COST-GED combines global sampling with local perturbation-based reconstruction. Global sampling generates complete matchings from scratch, while local reconstruction modifies the current matching and repairs conflicts.

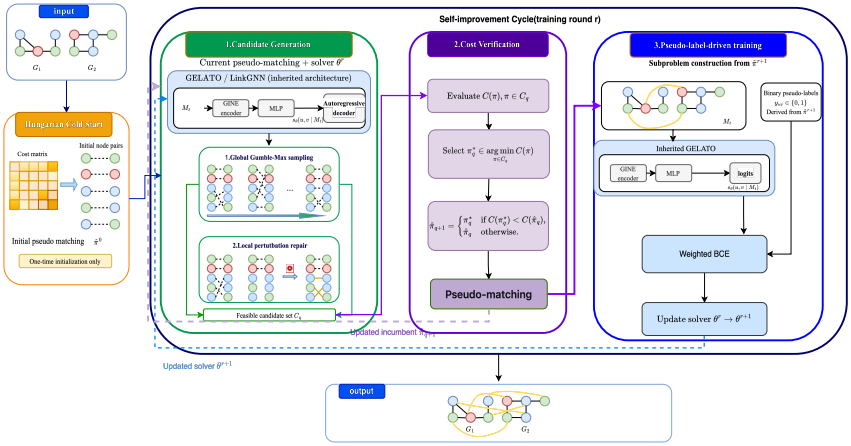


Figure 1: Overview of the COST-GED framework.

4.1 Candidate Generation

4.1.1 Hungarian Initialization

We initialize a feasible pseudo-matching for each training graph pair using the Hungarian algorithm Kuhn (1955) with a cost matrix based only on node labels. Since this cost matrix ignores edge structure, the resulting matching need not minimize the full edit cost. Subsequent self-improvement iterations refine these initial matchings through candidate generation and cost validation.

4.1.2 Candidate Matching Generator

We build on GELATO Pellizzoni et al. (2026), using its GINE-based LinkGNN as the candidate generator. At self-improvement iteration r, the model assigns scores $s_{\theta^r}(u, v \mid M_t)$ conditioned on the partial matching M_t before decoding step t. The unmatched node sets are $U_t = V_1 \setminus \mathrm{dom}(M_t)$ and $W_t = V_2 \setminus \mathrm{ran}(M_t)$, where $\mathrm{dom}(M_t)$ and $\mathrm{ran}(M_t)$ denote the already matched source and target nodes, respectively. Each decoding step selects a pair from $U_t \times W_t$ and updates the partial matching and candidate set.

We order each graph pair such that $|V_1| \leq |V_2|$ and decode until every source node is matched to a distinct target node; unmatched target nodes correspond to insertions. Model scores guide candidate generation, while the edit cost of each complete matching determines its quality.

4.1.3 Global Gumbel-Max Sampling

Global Gumbel-Max sampling broadens the search by generating complete matchings autoregressively from an empty matching. The model parameters θ^r remain fixed during candidate generation at iteration r. At decoding step t, the scores of candidate pairs $(u, v) \in U_t \times W_t$ define the sampling probabilities

$$p_{\theta^r}(u, v \mid M_t) = \frac{\exp(s_{\theta^r}(u, v \mid M_t)/\tau_g)}{\sum_{(u',v') \in U_t \times W_t} \exp(s_{\theta^r}(u', v' \mid M_t)/\tau_g)}. \tag{2}$$

Here, $p_{\theta^r}(u, v \mid M_t)$ is the probability of selecting (u, v) given M_t, and $\tau_g > 0$ is the global sampling temperature.

Sampling is implemented using the following Gumbel-Max rule for node-pair selection:

$$(u_t, v_t) = \underset{(u,v) \in U_t \times W_t}{\arg\max} \left[\frac{s_{\theta^r}(u, v \mid M_t)}{\tau_g} + g_{u,v} \right]. \tag{3}$$

At each decoding step, $g_{u,v}$ is independently sampled for every candidate pair from the standard Gumbel distribution with location 0 and scale 1. After selecting a pair, the partial matching and remaining candidate set are updated, and decoding continues until a complete matching is obtained.

To generate multiple candidates, the first decoding step selects the K node pairs with the highest Gumbel-perturbed scores as the starting points of K paths. Each path is then extended independently using the sampling rule above. After decoding, all candidate edit costs are evaluated, and the matching with the lowest cost is returned by the global sampling procedure. This matching is then passed to the cost-validation module described in Section 4.2.

The temperature τ_g controls sampling randomness: higher values flatten the probability distribution, while lower values concentrate it on high-scoring node pairs. We keep τ_g fixed throughout training; its value and sensitivity analysis are reported in the experimental section. This mechanism introduces stochastic exploration guided by model scores, providing diverse candidates for cost validation.

4.1.4 Local Perturbation and Repair

Global sampling generates alternative complete matchings from scratch, but this may discard useful correspondences already present in the current pseudo-matching. COST-GED therefore uses local perturbation and repair to construct new feasible candidates while retaining some existing correspondences.

Let $\hat{\pi}^r$ denote the current pseudo-matching at iteration r. During local reconstruction, the model separately generates K complete matching paths using the local temperature τ_l. The matching with the lowest edit cost, denoted by π_θ^r, supplies reference correspondences for local replacement. The local temperature is lower than the global temperature, concentrating sampling on node pairs with higher model scores. The specific parameter settings are provided in the experimental section.

The set of source nodes currently matched to non-null targets is $V_m^r = \{u \in V_1 \mid \hat{\pi}^r(u) \neq \epsilon\}$. When $|V_m^r| > 1$, we sample a perturbation set $D^r \subseteq V_m^r$ uniformly without replacement, with size $|D^r| = \max(1, \lfloor \rho |V_m^r| \rfloor)$, where $\rho \in (0, 1]$ is the perturbation ratio. Correspondences for nodes in D^r are replaced by those in the reference matching, while the remaining correspondences are retained, yielding

$$\bar{\pi}^r(u) = \begin{cases} \pi_\theta^r(u), & u \in D^r, \\ \hat{\pi}^r(u), & u \notin D^r. \end{cases} \tag{4}$$

When $|V_m^r| \leq 1$, we retain the current matching without applying any perturbation.

The mixed matching may contain conflicts in which multiple source nodes map to the same non-null target node. To restore injectivity, source nodes are processed in index order, retaining the first correspondence to each non-null target and mapping subsequent conflicting nodes to ϵ. This yields a feasible candidate $\tilde{\pi}^r$; conflict repair may also change correspondences outside the perturbation set.

Each local reconstruction call performs three independent trials of perturbation and repair using the same current pseudo-matching and the same newly generated reference matching. The candidate with the lowest edit cost is selected and passed to the cost-validation module described in Section 4.2.

4.2 Cost Validation

Global sampling and local reconstruction generate candidate matchings for each graph pair, but the scores produced by LinkGNN do not directly reflect the edit cost of a complete matching. Selecting a node pair affects both the remaining feasible correspondences and the resulting node and edge edit operations. Updating pseudo-matchings solely based on model scores may therefore increase the overall edit cost. COST-GED addresses this issue by decoupling candidate generation from candidate acceptance: the model proposes candidate matchings, while their edit costs determine whether they are accepted.

For a fixed graph pair (G_1, G_2), let q index successive candidate validations within self-improvement iteration r. Let $\hat{\pi}_q$ denote the pseudo-matching maintained before validation step

q, and write $C(\pi) = C_\pi(G_1, G_2)$ for brevity. Given a nonempty set of feasible candidates $\mathcal{C}_q$, we first select a minimum-cost candidate:

$$\pi_q^\star \in \arg\min_{\pi \in \mathcal{C}_q} C(\pi). \tag{5}$$

We then apply the following acceptance rule, which requires a strict cost reduction:

$$\hat{\pi}_{q+1} = \begin{cases} \pi_q^\star, & C(\pi_q^\star) < C(\hat{\pi}_q), \\ \hat{\pi}_q, & \text{otherwise.} \end{cases} \tag{6}$$

After all candidate validations in iteration r, the retained matching is denoted by $\hat{\pi}^{r+1}$ and used to construct supervision for the subsequent training stage.

Proposition 1 (Monotonicity of Pseudo-Matching Cost). For a fixed graph pair and a fixed edit-cost function, suppose that the initial pseudo-matching and all candidate matchings are feasible. Then the update rule satisfies

$$\mathrm{GED}(G_1, G_2) \le C(\hat{\pi}_{q+1}) \le C(\hat{\pi}_q), \qquad \forall\, q \ge 0. \tag{7}$$

Proof. If $\pi_q^\star$ is accepted, then $C(\pi_q^\star) < C(\hat{\pi}_q)$; otherwise, $\hat{\pi}_{q+1} = \hat{\pi}_q$. Hence, the maintained pseudo-matching cost is non-increasing. Because every retained matching remains feasible, its edit cost is no smaller than the minimum cost over all feasible edit paths, i.e., the exact GED. This proves both inequalities.

Proposition 1 guarantees that the pseudo-matching cost maintained for the same graph pair cannot increase across successive validations. COST-GED can therefore construct training feedback from cost-validated matchings without requiring exact GED values or optimal matchings as training supervision.

4.3 Iterative Training

After cost validation, the improved pseudo-matching $\hat{\pi}^{r+1}$ is converted into link-prediction subproblems following GELATO Pellizzoni et al. (2026). A subset of pseudo-matched correspondences is revealed to form a partial matching state M_t, while the remaining correspondences provide positive supervision for predicting unmatched node pairs. Details of negative sampling, instance augmentation, and subproblem construction are provided in Appendix A.

For each subproblem, LinkGNN encodes the joint graph with GINE and predicts pairwise matching probabilities:

$$s_\theta(u, v \mid M_t) = \mathrm{MLP}(h_u \,\|\, h_v), \qquad \widehat{y}_{uv} = \sigma(s_\theta(u, v \mid M_t)). \tag{8}$$

The solver is trained using weighted binary cross entropy:

$$\mathcal{L}_{\mathcal{B}} = -\frac{1}{|\mathcal{B}|} \sum_{(u,v)\in\mathcal{B}} \left[\alpha_{\mathcal{B}} y_{uv} \log \widehat{y}_{uv} + (1 - y_{uv}) \log(1 - \widehat{y}_{uv})\right], \tag{9}$$

where $y_{uv} \in \{0, 1\}$ denotes the pseudo-label and $\alpha_{\mathcal{B}} = N_{\mathcal{B}}^- / N_{\mathcal{B}}^+$ compensates for class imbalance.

After training, the updated solver is used to generate and validate candidates in the next self-improvement iteration, forming a cycle of candidate generation, cost validation, and model update. The training graph-pair pool is periodically refreshed, with newly sampled pairs initialized using the current solver. Exact GED values are used only for validation and model selection, and never enter pseudo-matching updates or the training loss.

5 Experiments

We implement COST-GED in PyTorch and evaluate it on an NVIDIA GeForce RTX 4090 GPU. Our experiments examine its performance on small-graph and larger-graph GED benchmarks, the effects of pseudo-matching feedback and global–local candidate generation, and the impact of training pool size and sampling settings on solution quality and computational efficiency.

5.1 Experimental Setup

Datasets and supervision. We use six small-graph benchmarks: AIDS, LINUX, and IMDB-16 Bai et al. (2019), ZINC-16 Gómez-Bombarelli et al. (2018), and MolHIV-16 and Code2-22 Hu et al. (2020). For these small-graph benchmarks, the suffix denotes the maximum number of nodes in either graph of a sampled graph pair. Graphs are split into training, validation, and test sets in a 6:2:2 ratio, with graph pairs constructed independently within each split.

Metrics. For N test pairs with exact GED d_i and output $\hat{d}_i$, we report normalized mean absolute error $\mathrm{nMAE} = \frac{100}{N}\sum_{i=1}^{N}\frac{|\hat{d}_i - d_i|}{\max(1,d_i)}$ and exact hit rate $\mathrm{EHR} = \frac{100}{N}\sum_{i=1}^{N}\mathbb{I}[\hat{d}_i = d_i]$, both as percentages. For matching-based methods, $\hat{d}_i$ is the cost of the returned feasible matching; otherwise, it is the predicted distance. EHR measures whether the returned cost reaches the exact GED, not whether a particular optimal matching is recovered. On larger graphs, we report the mean feasible-matching cost $\overline{C} = \frac{1}{N}\sum_{i=1}^{N} C_i$ and percentage reductions relative to explicitly identified baselines.

COST-GED budgets. The default small-graph configuration uses a periodically refreshed pool of 5,000 unlabeled graph pairs and $K = 32$ paths for both global and local candidate generation. Test inference uses $K = 32$ paths per decoding round over eight rounds and returns the lowest-cost matching found. Full training and decoding settings are provided in Appendix C, and analyses of candidate path budget, inference cost, and sampling parameters are presented in Appendix F.

Baselines and comparison protocol. We compare COST-GED with exact solvers, non-learning baselines, and learning-based or hybrid GED methods. The exact baseline is MIP-F2 Lerouge et al. (2017). Non-learning baselines include Hungarian Kuhn (1955), Refine Zeng et al. (2009), Branch Blumenthal & Gamper (2018), App-BMao Xu & Chang (2025), and IPFP Bougleux et al. (2017). Learning-based and hybrid baselines include GREED Ranjan et al. (2022), GELATO Pellizzoni et al. (2026), GEDGNN Piao et al. (2023), MATA* Liu et al. (2023), GraphEdX Jain et al. (2024), GEDHOT Cheng et al. (2025), GEDRanker Huang et al. (2025a), and GRAIL Verma et al. (2025). Reproduced methods use the same data splits. For methods that return explicit matchings, the costs of the returned matchings are recomputed under the common edit-cost function in Section 3; results taken from original papers are explicitly marked. In baseline evaluation, we run MIP-F2 with a 0.1 s time limit per pair and use the best feasible incumbent found within the limit. This time limit applies only to baseline evaluation, not to exact-reference generation. Detailed baseline sources, adaptations, and configurations are provided in Appendix D.

5.2 Main Results

Table 1(a) compares COST-GED with representative baselines on six small-graph GED benchmarks. With an active training pool of 5,000 unlabeled graph pairs, COST-GED matched or exceeded the EHR of the supervised baselines on AIDS, LINUX, ZINC-16, and MolHIV-16, while remaining within 0.1 percentage point on IMDB-16. The gains were particularly pronounced on ZINC-16 and MolHIV-16, supporting the effectiveness of cost-validated self-improvement on labeled molecular graphs. Performance on AIDS, LINUX, and IMDB-16 was close to saturation, leaving limited room for further improvement. Code2-22, which has the largest number of node label types (86), remained more challenging: with a 5,000-pair training pool, COST-GED still trailed the supervised baselines. Increasing the unlabeled training pool to 40K pairs raised EHR to 95.8%; the full pool-size analysis is provided in Appendix E.

Table 1(b) further compares COST-GED with GEDGW and GEDRanker, which reduce dependence on exact GED supervision. GEDGW Cheng et al. (2025) uses an optimal-transport formulation incorporating Gromov–Wasserstein discrepancy, while GEDRanker Huang et al. (2025a) learns relative preferences among candidate matchings from edit-path lengths. In contrast, COST-GED directly uses the computed edit costs of complete feasible matchings to validate pseudo-matching updates and construct training supervision, without a learned preference discriminator.

5.3 Evaluation on Larger Graphs

To evaluate COST-GED when exact supervision is difficult to obtain, we conducted experiments on MolHIV-30-50 and MolHIV-50-70. For each benchmark, all methods were evaluated on the

Table 1: Results on the small-graph GED benchmarks. (a) Main comparison on six benchmarks. (b) Comparison with methods that do not require exact GED values or optimal matchings as training supervision.

(a) main-results

Method	AIDS		LINUX		IMDB-16		ZINC-16		MolHIV-16		Code2-22	
	nMAE ↓	EHR ↑	nMAE ↓	EHR ↑	nMAE ↓	EHR ↑	nMAE ↓	EHR ↑	nMAE ↓	EHR ↑	nMAE ↓	EHR ↑
MIP-F2*	13.0±0.9	63.2±1.4	0.3±0.1	99.5±0.1	25.3±4.8	78.9±1.4	43.7±0.7	6.1±0.7	24.1±0.5	37.3±0.6	20.1±1.0	41.2±2.4
Branch	99.0±2.6	1.9±0.3	63.4±3.3	51.1±1.8	11.1±2.3	64.1±1.2	100.5±1.1	1.0±0.3	62.9±0.8	0.4±0.1	79.0±1.5	7.7±0.9
Refine	38.3±1.8	20.0±1.0	56.2±2.9	56.6±1.5	9.0±2.1	99.4±0.5	52.0±3.5	1.4±0.2	26.9±1.3	6.2±0.7	102.1±13.6	3.6±0.5
App-BMao	5.4±0.2	74.2±0.5	0.6±0.2	98.1±0.5	0.1±0.0	99.6±0.2	35.8±0.2	10.0±0.1	17.4±0.3	27.7±0.6	43.8±0.3	22.5±1.4
GREED	10.3±0.2	38.4±2.3	11.4±0.3	55.6±2.0	4.7±0.3	64.6±1.6	9.7±0.2	24.7±1.4	9.9±0.4	23.9±1.3	8.4±0.3	36.7±1.6
GraphEDX	8.0±0.3	51.5±0.6	13.8±0.4	81.4±0.4	12.7±1.3	49.2±1.8	7.7±0.2	36.3±1.1	9.9±0.3	21.9±0.9	6.0±0.8	69.3±1.3
MATA*	6.5±0.7	66.1±2.6	3.7±0.4	89.8±0.9	0.1±0.1	96.2±0.4	15.1±0.6	10.7±0.9	14.1±0.1	35.5±0.6	5.1±0.2	61.1±0.9
GEDGNN	21.1±0.8	38.7±0.9	6.7±0.6	85.8±0.9	2.3±0.5	94.1±0.7	46.6±1.1	3.5±0.6	32.1±0.2	4.8±0.7	17.0±1.0	64.2±1.2
GEDHOT	6.6±0.3	66.8±1.7	0.6±0.1	98.2±0.4	0.2±0.1	96.4±0.4	27.5±0.6	7.1±0.3	22.5±0.4	9.9±0.4	9.0±0.4	54.6±0.9
GRAIL	2.7±0.1	82.1±0.5	**0.0±0.0**	**100.0±0.0**	**0.0±0.0**	**99.9±0.1**	12.7±0.4	21.3±0.9	8.5±0.2	33.1±0.9	5.8±0.3	60.4±0.8
GELATO	0.1±0.0	99.3±0.3	0.1±0.1	99.9±0.1	0.1±0.3	**99.9±0.1**	0.7±0.1	91.1±1.1	0.5±0.4	95.3±0.8	**0.6±0.4**	**95.7±0.8**
COST-GED	**0.0±0.0**	**99.8±0.1**	**0.0±0.0**	**100.0±0.0**	**0.0±0.0**	99.8±0.1	**0.2±0.0**	**97.2±0.3**	**0.1±0.0**	**98.6±0.3**	1.0±0.2	89.9±1.7

*MIP-F2 is evaluated with a 0.1-second time limit per graph pair.

(b) Comparison without exact training supervision

Method	AIDS					LINUX					IMDB-16				
	MAE ↓	Acc(5%) ↑	ρ ↑	τ ↑	Time(s) ↓	MAE ↓	Acc(5%) ↑	ρ ↑	τ ↑	Time(s) ↓	MAE ↓	Acc(5%) ↑	ρ ↑	τ ↑	Time(s) ↓
GEDGW	0.848	52.7	0.857	0.768	**35.24**	0.491	76.4	0.916	0.864	**23.53**	0.374	94.2	0.969	0.955	**38.49**
GEDRanker	0.088	92.6	0.984	0.969	75.9	**0.000**	99.5	0.997	0.995	60.7	1.019	84.0	0.999	0.990	151.1
COST-GED (Ours)	**0.001**	**99.9**	**1.000**	**1.000**	97.2	**0.000**	**100.0**	**1.000**	**1.000**	65.2	**0.001**	**100.0**	**1.000**	**1.000**	132.3

Table 2: Mean edit cost of feasible matchings on the larger MolHIV benchmarks. Lower is better.

Method	**MolHIV-30-50 ↓**	**MolHIV-50-70 ↓**
Hungarian (node-label init.)	64.556	101.474
Hungarian	70.418	112.710
IPFP	44.914	71.320
COST-GED ($K = 32$)	**37.316**	**57.586**

same 500 test graph pairs under a common edit-cost function. Because exact GED references were unavailable, we compared the mean edit costs of the returned feasible matchings. As shown in Table 2, COST-GED achieved the lowest mean edit cost among the compared methods on both benchmarks. Relative to the Hungarian initialization based only on node labels, COST-GED reduced the mean edit cost by 42.20% on MolHIV-30-50 and 43.25% on MolHIV-50-70. Compared with IPFP, the best-performing traditional baseline in our comparison, the corresponding reductions were 16.92% and 19.26%, respectively. These results show that COST-GED can produce lower-cost feasible matchings on the tested larger graphs without exact GED values or optimal matchings as training supervision.

5.4 Ablation Studies

Table 3 evaluates pseudo-matching feedback and the candidate-generation strategy on ZINC-16 and Code2-22. For the feedback ablation, we disabled graph-pair pool refresh and varied only whether the training supervision was updated using improved pseudo-matchings. For the candidate-generation ablation, we retained pseudo-matching feedback and pool refresh, while replacing local reconstruction with global resampling under the same neural decoding path budget.

Enabling pseudo-matching feedback increased EHR by 83.0 and 56.9 percentage points on ZINC-16 and Code2-22, respectively, compared with retaining the initial labels, supporting the effectiveness of feeding improved matchings back into training. Compared with global resampling alone, combining global sampling with local reconstruction increased EHR by 0.6 and 1.4 percentage points on ZINC-16 and Code2-22, respectively, while also reducing nMAE on both datasets.

5.5 Pseudo-Matching Quality during Training

Figure 2 tracks pseudo-matching quality during training. At cycle 25, the mean normalized cost gaps on ZINC-16, Code2-22, and MolHIV-16 were 0.5814%, 0.7901%, and 0.3239%, respectively,

Table 3: Ablation study on pseudo-matching feedback and candidate generation. Each entry reports nMAE (%) ↓ / EHR (%) ↑.

Ablation	Setting	ZINC-16	Code2-22
Pseudo-matching feedback	Fixed initial training labels	36.2537 / 2.30	20.5393 / 25.90
	Updated training labels	**1.1722 / 85.30**	**1.7925 / 82.80**
Candidate generation	Global sampling only	0.2380 / 96.50	1.1535 / 88.80
	Global sampling + local reconstruction	**0.2137 / 97.10**	**0.8942 / 90.20**

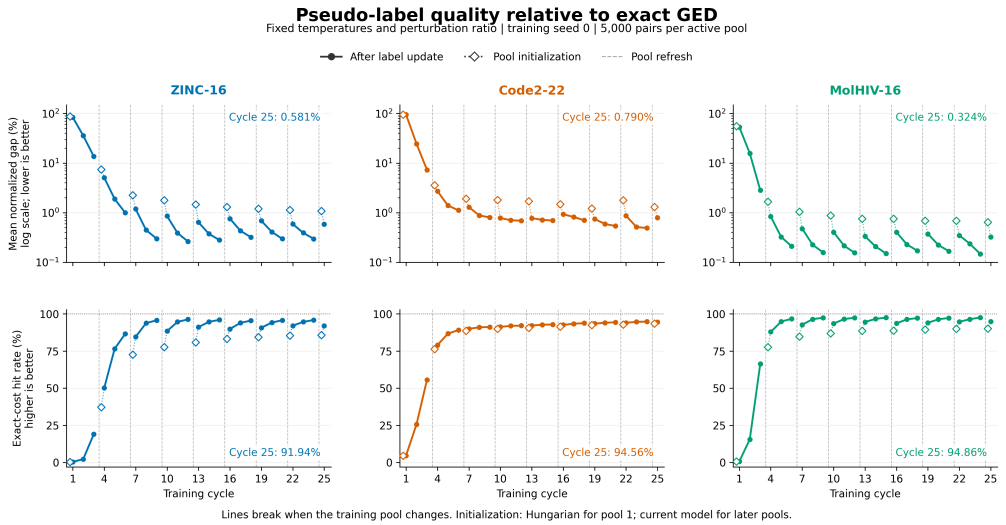


Figure 2: Pseudo-matching quality during training on ZINC-16, Code2-22, and MolHIV-16.

showing that the maintained pseudo-matchings can be progressively refined using computable edit costs. Consistent with Proposition 1, pseudo-matching costs are non-increasing within each fixed training pool. Because the pool is periodically refreshed, however, trajectories across refreshes involve different graph pairs and need not be globally monotonic.

6 Conclusion

We presented COST-GED, a cost-validated self-improvement framework for learning GED without exact GED values or optimal matchings as training supervision. It accepts only candidates with strictly lower edit costs and feeds improved matchings back into training, guaranteeing non-increasing pseudo-matching costs for each fixed graph pair and edit-cost function. Global sampling and local perturbation-based reconstruction combine exploration with reuse of existing correspondences to mitigate search stagnation under a limited budget. Across six public small-graph GED benchmarks, COST-GED achieves performance competitive with supervised methods. Feedback ablations and pseudo-matching quality analyses further support the effectiveness of cost-based training feedback. On MolHIV-30-50 and MolHIV-50-70, it reduces the mean edit cost of feasible matchings by 16.92% and 19.26%, respectively, relative to IPFP, the best-performing traditional baseline in our comparison. Future work will explore more efficient candidate generation and inference, alongside integration with traditional search and pruning strategies, to improve scalability.

AI use statement

In this work, we used generative AI tools to assist with English language editing, rewriting for clarity, LaTeX formatting, and code modification and debugging. All AI-assisted text and code were reviewed, revised, and verified by the authors. Experimental results, analyses, technical claims, and final scientific conclusions were independently checked by the authors. We take full responsibility

for the final content of this work, including text, code, claims, and artifacts produced with the aid of generative AI.

Reproducibility statement

We provide the model architecture, training procedure, candidate-generation settings, inference configuration, evaluation metrics, dataset construction, and data split protocols in Sections 4 and 5. Additional hyperparameter sensitivity results and implementation details are provided in the appendix. All experimental settings are specified to support reproduction of the reported results. We plan to release the code and configuration files upon publication.

References

Y. Bai, H. Ding, S. Bian, T. Chen, Y. Sun, and W. Wang. SimGNN: A neural network approach to fast graph similarity computation. In *Proceedings of the 12th ACM International Conference on Web Search and Data Mining*, pp. 384–392, 2019.

Y. Bai, H. Ding, K. Gu, Y. Sun, and W. Wang. Learning-based efficient graph similarity computation via multi-scale convolutional set matching. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 3219–3226, 2020.

D. B. Blumenthal and J. Gamper. Improved lower bounds for graph edit distance. *IEEE Transactions on Knowledge and Data Engineering*, 30(3):503–516, 2018.

D. B. Blumenthal and J. Gamper. On the exact computation of the graph edit distance. *Pattern Recognition Letters*, 134:46–57, 2020.

S. Bougleux, L. Brun, V. Carletti, P. Foggia, B. Gaüzère, and M. Vento. Graph edit distance as a quadratic assignment problem. *Pattern Recognition Letters*, 87:38–46, 2017.

L. Chang, X. Feng, X. Lin, L. Qin, W. Zhang, and D. Ouyang. Speeding up GED verification for graph similarity search. In *Proceedings of the 36th IEEE International Conference on Data Engineering (ICDE)*, pp. 793–804, 2020.

Q. Cheng, D. Yan, T. Wu, Z. Huang, and Q. Zhang. Computing approximate graph edit distance via optimal transport. *Proceedings of the ACM on Management of Data*, 3(1):23:1–23:26, 2025.

C. Garcia-Hernandez, A. Fernández, and F. Serratosa. Ligand-based virtual screening using graph edit distance as molecular similarity measure. *Journal of Chemical Information and Modeling*, 59(4):1410–1421, 2019.

R. Gómez-Bombarelli, J. N. Wei, D. Duvenaud, J. M. Hernández-Lobato, B. Sánchez-Lengeling, D. Sheberla, J. Aguilera-Iparraguirre, T. D. Hirzel, R. P. Adams, and A. Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *ACS Central Science*, 4(2):268–276, 2018.

W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec. Open graph benchmark: Datasets for machine learning on graphs. In *Advances in Neural Information Processing Systems*, volume 33, pp. 22118–22133, 2020.

W. Huang, H. Wang, D. Wen, S. Ma, W. Zhang, and X. Lin. Towards unsupervised training of matching-based graph edit distance solver via preference-aware GAN. In *Advances in Neural Information Processing Systems*, volume 38, pp. 178996–179023, 2025a.

W. Huang, H. Wang, D. Wen, W. Zhang, Y. Zhang, and X. Lin. DiffGED: Computing graph edit distance via diffusion-based graph matching. *arXiv preprint arXiv:2503.18245*, 2025b.

E. Jain, I. Roy, S. Meher, S. Chakrabarti, and A. De. Graph edit distance with general costs using neural set divergence. In *Advances in Neural Information Processing Systems*, volume 37, pp. 73399–73438, 2024.

Z. Jia, Y. Xiong, Y. Nan, Y. Zhang, J. Zhao, and M. Wen. MAGIC: Detecting advanced persistent threats via masked graph representation learning. In *33rd USENIX Security Symposium (USENIX Security 24)*, pp. 5197–5214. USENIX Association, 2024.

D. Justice and A. Hero. A binary linear programming formulation of the graph edit distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(8):1200–1214, 2006.

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

J. Lerouge, Z. Abu-Aisheh, R. Raveaux, P. Héroux, and S. Adam. New binary linear programming formulation to compute the graph edit distance. *Pattern Recognition*, 72:254–265, 2017.

Y. Li, C. Gu, T. Dullien, O. Vinyals, and P. Kohli. Graph matching networks for learning the similarity of graph structured objects. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 3835–3845. PMLR, 2019.

J. Liu, M. Zhou, S. Ma, and L. Pan. MATA*: Combining learnable node matching with A* algorithm for approximate graph edit distance computation. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, pp. 1503–1512, 2023.

Paolo Pellizzoni, Till Schulz, and Karsten Borgwardt. Gelato: Graph edit distance via autoregressive neural combinatorial optimization. In *International Conference on Learning Representations*, 2026.

C. Piao, T. Xu, X. Sun, Y. Rong, K. Zhao, and H. Cheng. Computing graph edit distance via neural graph matching. *Proceedings of the VLDB Endowment*, 16(8):1817–1829, 2023.

J. Pirnay and D. G. Grimm. Self-improvement for neural combinatorial optimization: Sample without replacement, but improvement. *Transactions on Machine Learning Research*, 2024.

R. Ranjan, S. Grover, S. Medya, V. Chakaravarthy, Y. Sabharwal, and S. Ranu. GREED: A neural framework for learning graph distance functions. In *Advances in Neural Information Processing Systems*, volume 35, pp. 22518–22530, 2022.

P. Riba, A. Fischer, J. Lladós, and A. Fornés. Learning graph edit distance by graph neural networks. *Pattern Recognition*, 120:108132, 2021.

S. Verma, A. Goyal, A. Mathur, A. Anand, and S. Ranu. GRAIL: Graph edit distance and node alignment using LLM-generated code. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 61301–61322. PMLR, 2025.

M. Xu and L. Chang. Graph edit distance estimation: A new heuristic and a holistic evaluation of learning-based methods. *Proceedings of the ACM on Management of Data*, 3(3):167:1–167:24, 2025.

L. Yang and L. Zou. Noah: Neural-optimized A* search algorithm for graph edit distance computation. In *Proceedings of the 37th IEEE International Conference on Data Engineering (ICDE)*, pp. 576–587, 2021.

Z. Zeng, A. K. H. Tung, J. Wang, J. Feng, and L. Zhou. Comparing stars: On approximating graph edit distance. *Proceedings of the VLDB Endowment*, 2(1):25–36, 2009.

A Additional Training Details

Subproblem construction. Given an updated pseudo-matching $\hat{\pi}^{r+1}$, we randomly select a subset of source nodes and reveal their assignments. Revealed non-null correspondences define the partial matching M_t and are represented as cross-graph alignment edges in the joint graph. The remaining assignments provide supervision for predicting unrevealed correspondences. Sampling revealed subsets of different sizes produces multiple training subproblems from the same pseudo-matching.

Table 4: Statistics of the datasets used in our experiments. Node and edge labels denote the numbers of distinct label types.

Dataset	#Graphs	Avg. Nodes	Avg. Edges	Node Labels	Edge Labels
AIDS	700	8.9	8.8	29	1
LINUX	1,000	7.6	6.9	1	1
IMDB	1,185	9.6	33.4	1	1
ZINC	7,489	20.4	21.7	18	3
MolHIV	23,687	18.5	19.8	51	4
Code2	6,183	24.9	23.9	86	1
MolHIV-30-50	8,787	36.8	39.2	44	4
MolHIV-50-70	1,179	57.6	59.8	33	4

Positive and negative samples. For each unrevealed source node u with $\hat{\pi}^{r+1}(u) \neq \epsilon$, the pair $(u, \hat{\pi}^{r+1}(u))$ is a positive sample. Candidate negative pairs connect these source nodes to remaining available target nodes other than their assigned targets. We apply GELATO's filtering rule based on Weisfeiler–Leman (WL) structural features to select negative samples Pellizzoni et al. (2026). Samples from different subproblems are counted separately, even when they share the same node indices, because they are conditioned on different partial matchings.

Deletion markers. An assignment $\hat{\pi}^{r+1}(u) = \epsilon$ represents a node deletion rather than a source–target link. Revealing such an assignment therefore adds no alignment edge. Unrevealed source nodes assigned to ϵ are excluded from both positive and negative link samples; deletion markers are not treated as target nodes for link prediction.

Node reindexing. We augment the training instances by randomly permuting source node indices. Source node features, edge indices, and pseudo-matching assignments are updated consistently, while the target graph remains unchanged. This preserves the graph structure and matching correspondences while varying the source node ordering used to construct training subproblems.

B Datasets and Evaluation Protocols

Table 4 summarizes the dataset statistics.

C Implementation Details

Model architecture. COST-GED uses a GINE encoder with five layers and a hidden dimension of 128. The matching head is an MLP with dimensions $256 \to 128 \to 1$, applied to the concatenated source and target node representations.

Training configuration. For the small-graph experiments with fixed sampling parameters, we use AdamW with an initial learning rate of 10^{-3} and zero weight decay. The learning rate is multiplied by 0.85 after each self-improvement cycle. Training runs for 25 cycles, with 10 epochs per cycle. Each graph pair yields 40 training subproblems, and each training batch contains 256 subproblems.

Training pool and candidate generation. The training graph-pair pool is refreshed every three cycles. Pseudo-matchings for newly sampled graph pairs are initialized using the current solver. Candidate generation uses $K = 32$ paths per call, with global temperature $\tau_g = 0.5$, local temperature $\tau_l = 0.25$, and perturbation ratio $\rho = 0.3$. These parameters remain fixed during training. The candidate-generation batch size is 32 graph pairs.

Validation and model selection. We select the checkpoint with the lowest validation nMAE. Validation uses $K = 32$ paths per decoding round and four rounds: one deterministic round followed by three Gumbel-sampling rounds at temperature 0.3. For each graph pair, the lowest-cost matching across all rounds is retained.

Table 5: A*+LSa results with a time limit of 10 seconds per graph pair. Completion denotes the percentage of pairs solved exactly. Mean runtime is computed only over exactly solved pairs, excluding timed-out pairs. Dashes indicate that full-test-set nMAE and EHR are not reported because some pairs time out.

Dataset	Completion	Timeouts	nMAE (%) ↓	EHR (%) ↑	Mean time on solved pairs (ms/pair)
AIDS	100%	0	0.0000	100.00	18.37
LINUX	100%	0	0.0000	100.00	18.72
IMDB-16	98.6%	14	—	—	365.52
ZINC-16	100%	0	0.0000	100.00	68.64
MolHIV-16	99.9%	1	—	—	80.12
Code2-22	95.1%	49	—	—	614.94

Table 6: Effect of active training pool size on Code2-22. Pool size denotes the number of graph pairs maintained at a time, rather than the total number of distinct pairs seen throughout training.

Training pool size	2K	5K	10K	20K	40K
EHR (%) ↑	84.5	89.9	90.7	92.1	95.8

Test-time decoding. Testing uses $K = 32$ paths per decoding round and eight rounds: one deterministic round followed by seven Gumbel-sampling rounds at temperature 0.5. The matching with the lowest edit cost across all rounds is returned. The test batch size is 32 graph pairs.

D Baseline Configurations

MIP-F2 Lerouge et al. (2017) and A*+LSa Chang et al. (2020) are run with time limits of 0.1 and 10 seconds per graph pair, respectively. MIP-F2 returns the best feasible solution found within its budget; this time-limited evaluation is distinct from exact-reference generation. For A*+LSa, we report the exact-solution completion rate and record timed-out pairs separately (Table 5). All methods that we reproduce are retrained or evaluated using the same data splits. For methods returning feasible matchings, we recompute edit costs under our common cost model: node and edge insertions and deletions cost 1, while substitutions cost 0 for identical labels and 1 otherwise. Results from methods that we cannot fully reproduce are taken from their original papers and explicitly marked in the comparison tables.

E Effect of Training Pool Size

Code2-22 is the most challenging of the six small-graph benchmarks and has the largest number of node label types (86). We evaluate COST-GED with active training pools of 2K, 5K, 10K, 20K, and 40K unlabeled graph pairs. As shown in Table 6, EHR improves consistently with pool size, increasing from 84.5% at 2K to 95.8% at 40K, comparable to GELATO trained on 5K supervised pairs (95.7%). These results show that COST-GED benefits from a larger training pool on this label-diverse benchmark. Expanding the pool requires no additional exact GED values or optimal matchings as training supervision, avoiding the associated label-generation cost.

F Hyperparameter Sensitivity and Inference Cost

Multi-round Gumbel sampling and inference cost. Multiple rounds of Gumbel sampling increase the diversity of candidate matchings, but also introduce additional inference cost. As shown in Table 7, increasing the number of candidate-generation paths K generally improves EHR, while inference time grows approximately linearly. When $K > 32$, the performance gains become progressively smaller. We therefore use $K = 32$ as a compromise between solution quality and inference cost.

Table 7: Effect of the number of sampled paths K on EHR and inference time.

	ZINC-16		Code2-22	
K	EHR (%) ↑	Time (s) ↓	EHR (%) ↑	Time (s) ↓
1	50.6 ± 0.9	7.2	66.6 ± 1.2	9.0
4	73.7 ± 1.1	7.8	74.8 ± 1.2	9.7
8	82.8 ± 1.0	8.0	78.6 ± 1.3	10.8
16	89.9 ± 0.7	9.6	83.9 ± 1.1	14.4
32	94.7 ± 0.5	13.5	88.5 ± 0.8	24.0
64	$\mathbf{97.8 \pm 0.6}$	23.4	$\mathbf{91.4 \pm 0.7}$	46.1

Table 8: Sensitivity to global temperature τ_g, local temperature τ_l, and perturbation ratio ρ on ZINC-16.

τ_g	τ_l	ρ	nMAE (%) ↓	EHR (%) ↑
0.3	0.15	0.30	0.3047	95.60
0.5	0.25	0.30	0.2137	97.10
0.9	0.45	0.30	0.2198	96.80
0.5	0.25	0.15	0.1496	97.70
0.5	0.25	0.50	0.2618	96.20

Sampling temperatures and perturbation ratio. We further examine the sensitivity of the global sampling temperature τ_g, local sampling temperature τ_l, and local perturbation ratio ρ on ZINC-16. The results show that moderate sampling temperatures provide a good balance between exploration and candidate quality, whereas an excessively large perturbation ratio can degrade matching quality. We use $(\tau_g, \tau_l, \rho) = (0.5, 0.25, 0.3)$ as the default configuration, selected according to validation performance. The complete sensitivity results are reported in Table 8.