Streamlining Computational Lithography With Efficient Pattern Database

Su Zheng, Wenqian Zhao, Shuyuan Sun, Fan Yang, *Member, IEEE*, Bei Yu, *Senior Member, IEEE*, and Martin D. F. Wong, *Life Fellow, IEEE*

***Abstract*—In the pursuit of advancing computational lithography, this article introduces a novel pattern database framework designed to support related tasks. The proposed framework is built upon three core components: 1) an unsupervised metric learning method for robust pattern embedding; 2) a vector database for swift pattern retrieval; and 3) an efficient algorithm dedicated to pattern clustering. These elements synergize to significantly enhance the efficiency and effectiveness of various computational lithography methods. In downstream tasks, our framework provides accurate lithography hotspot detection through pattern retrieval, streamlines inverse lithography technique (ILT) by leveraging solution reusing, and facilitates the exploration of ILT & source parameters based on the pattern clustering results. Collectively, these advancements culminate in a comprehensive improvement in computational lithography, offering a scalable solution for the ever-evolving demands of this field.**

***Index Terms*—Lithography hotspot detection (HSD), pattern clustering, pattern database.**

I. Introduction

THE FIELD of computational lithography has been a driving force behind the rapid advancement of semiconductor technology, facilitating the production of ever-smaller and more powerful electronic devices. As the industry pushes toward the limits of Moore's Law, the challenges associated with pattern miniaturization have become increasingly pronounced. To overcome these obstacles, mask optimization methods, such as optical proximity correction (OPC) [1], [2], [3], inverse lithography technique (ILT) [4], [5], [6], [7], [8], and their machine-learning extensions [9], [10], [11], [12], [13], [14] have been developed to improve the printability of fine features. While ILT has shown promise in enhancing pattern fidelity, it is characterized by high computational requirements, making it a time-intensive endeavor that often becomes a bottleneck in production.

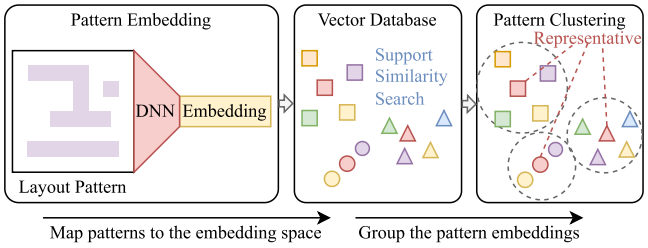

Fig. 1. Components of the pattern database, which include the unsupervised metric learning for pattern embedding, the vector database for pattern retrieval, and the pattern clustering algorithm for representative pattern selection.

In the mask optimization flow, hotspot detection (HSD) algorithms are employed to identify regions prone to errors during the lithography process, which will be refined by more intricate and computationally intensive OPC/ILT recipes. Current HSD methods usually rely on pattern matching [15], [16], [17] or deep neural networks (DNNs) [18], [19], [20], [21], [22]. Although DNN-based methods stand out by their high accuracy, due to the long training time and unsatisfactory generalization ability of deep learning, these methods usually fall short of accuracy on unseen patterns.

In this article, we design a pattern database to address key challenges in computational lithography, including ILT, HSD, and related applications. For instance, the generalization problem in DNN-based HSD can be mitigated through an effective pattern matching method built upon our elastic pattern database. Moreover, given the fact that ILT is a computationally intensive process, our pattern database can accelerate it by enabling the reuse of ILT results for similar patterns. As noted in [23], reusing ILT solutions can reduce the time for mask optimization. Alongside ILT, optimizing source parameters is another crucial aspect of computational lithography [24]. This process is similarly time-intensive, requiring extensive simulations to determine the optimal source configuration. By employing pattern clustering to identify representative patterns, the optimization process can focus on refining source parameters for these key patterns, thereby reducing the overall computational burden. A similar strategy can also be applied to ILT parameter optimization, refining critical factors like the step size in gradient descent.

In essence, a pattern database can address the shortcomings of current computational lithography by providing a more efficient and accurate method for pattern analysis, HSD,

Received 2 December 2024; revised 26 February 2025; accepted 11 April 2025. Date of publication 17 April 2025; date of current version 22 October 2025. This work was supported in part by the Research Grants Council of Hong Kong, SAR, under Grant RFS2425-4S02; in part by the MIND Project under Grant MINDXZ202404; and in part by the National Key R&D Program of China under Grant 2020YFA0711900 and Grant 2020YFA0711903. This article was recommended by Associate Editor Y. Lin. *(Corresponding author: Bei Yu.)*

Su Zheng, Wenqian Zhao, and Bei Yu are with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong, SAR (e-mail: szheng22@cse.cuhk.edu.hk; byu@cse.cuhk.edu.hk).

Shuyuan Sun and Fan Yang are with the State Key Laboratory of Integrated Circuits and Systems, Fudan University, Shanghai 200433, China.

Martin D. F. Wong is with the Department of Computer Science, Hong Kong Baptist University, Hong Kong, SAR.

Digital Object Identifier 10.1109/TCAD.2025.3562158

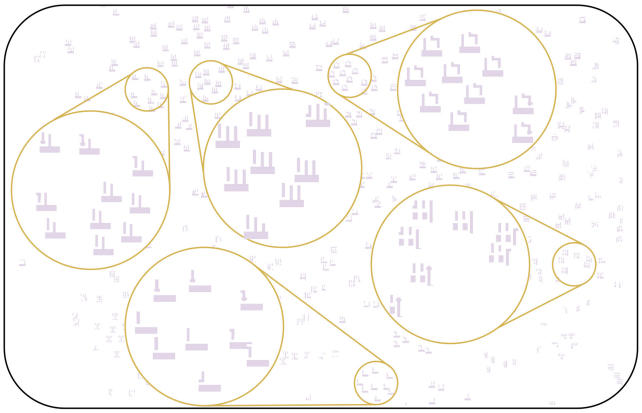

Fig. 2. Visualization of the embedding space learned by our pattern embedding method. Based on our unsupervised metric learning methodology, similar patterns can be clustered together, and dissimilar ones can be separated.

ILT solution reusing, and the optimization of ILT/source parameters. This leads to a more streamlined manufacturing process, capable of producing more reliable circuits in a shorter amount of time. As illustrated by Fig. 1, our pattern database is a comprehensive solution designed to address the challenges of computational lithography through three innovative components.

Unsupervised Metric Learning for Pattern Embedding: At the heart of the framework lies an unsupervised metric learning algorithm that projects layout patterns into a vector space where similar patterns are clustered together, and dissimilar ones are separated. As shown in Fig. 2, the embedding process facilitates the clustering of layout patterns, and enables a deeper understanding of the intrinsic characteristics of each pattern.

Vector Database for Pattern Retrieval: Complementing our pattern embedding method is a highly efficient vector database, which is optimized for high-speed retrieval of pattern data, ensuring that the most relevant patterns are accessible in real-time. This capability is crucial for adapting to the dynamic requirements of similarity search and solution reusing.

Efficient Algorithm for Pattern Clustering: A novel hierarchical K-neighbor graph partitioning method is proposed to cluster massive patterns into a specified number of groups. The center point of each group can become a representative point of other patterns in the same group. The pattern clustering algorithm can not only help users to analyze a huge number of patterns efficiently but also streamline lithography-related optimization processes by focusing computational resources on the most representative patterns.

Together, these components form a robust framework that enhances the detection of lithography hotspots, accelerates the ILT process, and enables the efficient exploration of ILT & source parameters on a curated set of representative patterns. With these powerful applications, our framework achieves a comprehensive improvement in computational lithography, paving the way for more efficient, accurate, and adaptable manufacturing processes. As illustrated by Fig. 3, the primary objective of this framework is to promote the development of computational lithography by enhancing efficiency, accuracy, and adaptability through the following goals.

Enhanced HSD: Utilizing the pattern embeddings, we identify hotspots by retrieving similar points in the pattern database. Through the effective embedding method, we can improve the HSD accuracy over existing image classification methods. Furthermore, the extensibility of pattern database makes it easier for our method to adapt to new hotspot data.

Accelerated ILT Process: We streamline ILT by employing a vector database for rapid pattern retrieval and solution reusing. This acceleration is intended to reduce the time and computational resources required for ILT, thereby enabling faster design iterations and time-to-market.

ILT/Source Parameter Exploration: We select representative patterns that are critical for ILT and source parameter optimization according to the pattern clustering results. By focusing on these key patterns, the proposed framework seeks to refine the lithography-related processes, leading to better quality and performance of the final semiconductor products.

As such, it holds the potential to significantly improve the efficiency, accuracy, and adaptability of computational lithography. The remainder of this article is organized as follows. Section II introduces the preliminaries of computational lithography and pattern database. Section III shows the details of the proposed methods. Section IV presents various experimental results that can prove the effectiveness of our framework. The conclusion is shown in Section V.

II. Preliminaries

A. Computational Lithography

Computational lithography has emerged as a pivotal technology in semiconductor manufacturing, including lithography simulation, ILT, etc. In this section, we will introduce the principles of lithography simulation and ILT, as well as their relationship to the applications of our pattern database.

Lithograph Simulation: The lithography process prints the patterns on the mask to the wafer. Given the input mask, lithography simulation predicts the printed patterns on the wafer. Ideally, the printed patterns are required to be as close as possible to our target patterns (i.e., shapes in the circuit layout). Measuring the difference between the printed patterns and our target patterns can estimate the quality of a mask, and provide information for mask optimization.

During the lithography process, the mask image M is first transformed through an optical projection system into the aerial image I. The distribution of light intensity at the wafer plane then forms the printed image Z with the development and etching processes. Lithography simulation typically consists of two parts: an optical projection model and a photoresist model. The Abbe and Hopkins models can be utilized in optical projection. In Abbe model, each element of the aerial image, $I(x, y)$, is obtained by

$$\int_{\sigma_x}\int_{\sigma_y}\left|M(x,y)\otimes\mathcal{F}^{-1}\{U(\sigma_x,\sigma_y)\}\right|^2 S(\sigma_x,\sigma_y)d\sigma_x d\sigma_y \quad (1)$$

where $\otimes$ represents convolution, $\mathcal{F}^{-1}$ denotes inverse FFT, U is the pupil function, and S is the source. Notations σ_x and σ_y represent the normalized spatial frequencies.

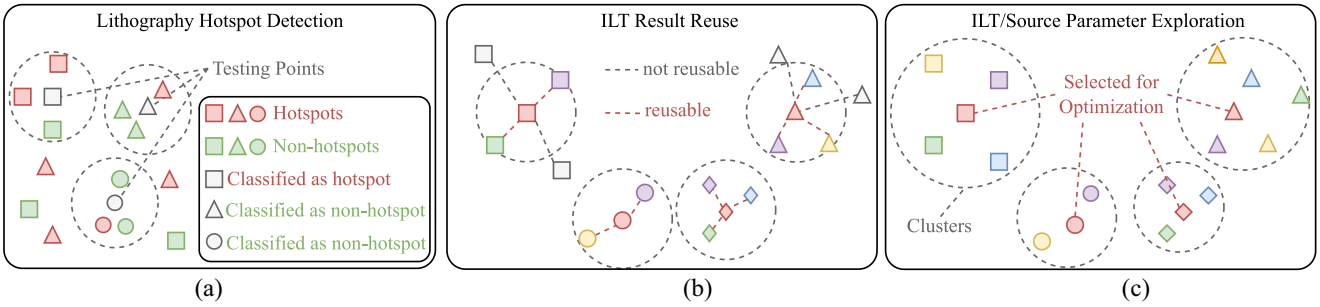

Fig. 3. Applications of the proposed pattern database. (a) HSD via pattern retrieval. A test sample is classified as a hotspot if the ratio of retrieved hotspots exceeds a predefined threshold. (b) ILT solution reusing, which can reduce the time and computational resources required for ILT. (c) ILT/source parameter exploration. Using representative points in the optimization ensures high efficiency and quality.

In Hopkins model, the aerial image I is computed by convolving the mask M with the kernels H, formulated by

$$I(x,y) = \sum_{k=1}^{N_h} w_k |M(x,y) \otimes h_k(x,y)|^2 \tag{2}$$

where N_h is the number of optical kernels, h_k is the k-th kernel of the optical kernel set H, and w_k is the weight of h_k. After optical simulation, the aerial image I is input into the photoresist model, which is a binarization function with an intensity threshold I_{th}. It outputs the printed image Z.

ILT: ILT's optimization objective is typically built on the printed image Z given by lithography simulation. The target of ILT is to find an optimized mask by $M^* = \text{ILT}(Z_t) = \min_M L(f(M), Z_t)$, where Z_t is the design target, $f(\cdot)$ stands for the lithography process, and $L(\cdot)$ is the loss function which typically considers the L2 loss and process variation band [4].

To optimize the mask image M, ILT starts with a matrix P, which can be converted to M by

$$\vec{M}(x,y) = \frac{1}{1+\exp\left(-\theta_M(P(x,y) \ - \gamma_M)\right)}. \tag{3}$$

The steepness θ_M and offset γ_M are two parameters that we can adjust during ILT parameter optimization.

To iteratively improve the mask, ILT can adopt loss functions that can optimize the evaluation metrics like $L2$ and PVB. For example

$$L\big(\vec{Z}_t, \vec{Z}\big) = \alpha \cdot L2\big(\vec{Z}_t, \vec{Z}\big) + \beta \cdot PVB\big(\vec{Z}_t, \vec{Z}\big). \tag{4}$$

The scales α and β are two parameters that we can adjust. The metric $L2$ calculates the difference between $\vec{Z}_t$ and $\vec{Z}$, while PVB (process variation band) evaluates the maximum discrepancy between the printed images under different process conditions. In addition, we also use the EPE metric to measure the edge placement errors (EPEs). For the details of these metrics, please refer to [25].

At each optimization step, the matrix P is updated by

$$P \leftarrow P - \eta \frac{\partial L(Z_t, Z)}{\partial P} \tag{5}$$

where η is the learning rate in gradient descent.

Relationship to Pattern Database: In our first application (i.e., HSD), the Hopkins model (2) is utilized to generate HSD labels. The ILT algorithm introduced in this section is used in our second application (i.e., accelerated ILT process). Moreover, the third application (i.e., ILT/source parameter optimization) is build upon the Abbe model (1) and the ILT algorithm.

B. Optimization of ILT and Source Parameters

In this section, we introduce the problem formulations of our third application (i.e., ILT/source parameter exploration). ILT algorithms involve multiple parameters that significantly influence the performance, such as the step size in gradient descent and the weights of different components in the loss function. However, these parameters are not refined in the normal ILT process, which indicates that the algorithm's performance cannot be assured. To conquer this challenge, we can utilize Bayesian optimization methods to find the optimal ILT parameters. Given N input masks, the optimization of parameters θ^*_{ILT} can be formulated as

$$\arg\min_{\theta_{\text{ILT}}} \sum_{n=1}^{N} L\big(f(\text{ILT}(Z_t^n|\theta_{\text{ILT}})), Z_t^n\big). \tag{6}$$

Specifically, the parameters θ^*_{ILT} can include the steepness θ_M and the offset γ_M in (3), the scales α and β in (4), and the step size in (5). The optimization algorithm we use will be introduced in Section III-F.

Utilizing the Abbe model in (1), the light source $S(\sigma_x, \sigma_y)$ can be optimized through gradient descent. Given N input masks $\{M^n|_{n=1}^N\}$, we can optimize the source S by

$$S^* = \arg\min_{S} \sum_{n=1}^{N} L\big(f(M^n|S), Z_t^n\big). \tag{7}$$

The details of our method will be introduced in Section III-F.

These optimization processes usually cost a large amount of time. Therefore, a pattern clustering method is required to find the most representative patterns in massive data, which can significantly simplify the optimization.

C. Vector Database

Vector database [26] is a specialized type of database designed to store, manage, and index high-dimensional vector data efficiently. Unlike traditional databases that deal with rows and columns of scalar values, vector databases handle vectors, which are the embeddings of complex objects,

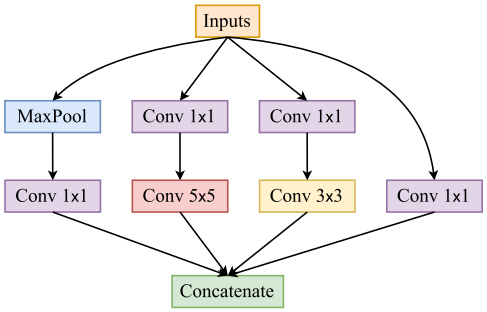

Fig. 4. Structure of the Inception block used in GoogLeNet. The multi-branch design of the Inception block allows GoogLeNet to learn diverse representations.

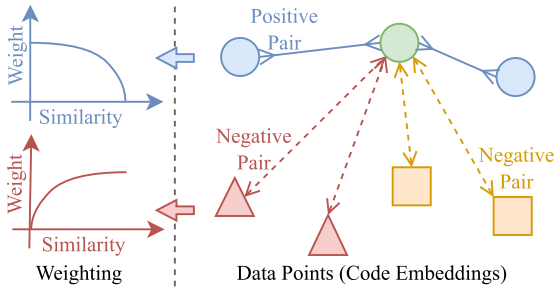

Fig. 5. In the weighting mechanism of MS loss, positive pairs with low similarity have higher weights, and so do negative pairs with high similarity.

such as words, images, videos, and audio. These vectors are generated by machine learning models and are used to capture the essential features of the data they represent. The efficiency of vector databases in handling large volumes of complex data makes them ideal for applications driven by artificial intelligence, including computational lithography. They support the rapid processing and analysis of data, which is essential for tasks, such as HSD and optimization of lithography processes. Various methodologies have been designed to efficiently manage and retrieve high-dimensional data, including the graph-based, hashing-based, quantization-based algorithms. Such algorithms are chosen based on their performance tradeoffs, which include factors like latency, accuracy, and domain-specific characteristics.

III. Method

In this section, we first discuss the methodologies employed in the proposed pattern database, which consists of three key components: 1) the pattern embedding model; 2) the vector database; and 3) the pattern clustering algorithm. Each of these components plays a crucial role in enabling efficient retrieval and analysis of layout patterns in computational lithography. The pattern embedding model is designed to extract representative features from layout patterns, facilitating accurate retrieval and clustering. The vector database enables efficient pattern retrieval. By retrieving previously encountered patterns, we can reuse their HSD/ILT results, which help improve accuracy and reduce computation time. The pattern clustering algorithm groups similar patterns together based on their embeddings. By organizing patterns into meaningful clusters, the algorithm aids in understanding variations within a layout and provides insights into optimal process parameters.

Next, we demonstrate how our pattern database can be adopted in three downstream tasks.

1) *Retrieval-Based HSD:* By leveraging the vector database's ability to retrieve previously identified patterns, HSD can be accelerated by matching new layouts against known problematic regions, reducing the need for exhaustive simulations.
2) *Solution Reusing in ILT:* ILT is computationally expensive, but with efficient retrieval of similar patterns, previously computed ILT solutions can be reused, significantly improving efficiency.
3) *ILT/Source Parameter Optimization:* The clustering algorithm enables users to identify the representative patterns from a large dataset, facilitating efficient ILT/source parameter optimization. By integrating these methodologies, the proposed pattern database provides a powerful toolset for computational lithography, offering both efficiency and scalability in handling complex tasks.

A. Pattern Embedding

In a pattern database, a model is trained to learn the similarities between various patterns through a process that focuses on the intrinsic characteristics of the patterns themselves, rather than any specific downstream applications. To this end, considering the vast repository of unlabeled layout patterns, we implement an unsupervised metric learning methodology for the purpose of pattern embedding. The unsupervised metric learning paradigm is adept at navigating the complexities of high-dimensional data, distilling it into a more manageable form without predefined labels.

Our embedding method outputs the vector representations of layout tiles. To extract the pattern features, we adopt GoogLeNet [27]. As illustrated by Fig. 4, the Inception block in GoogLeNet captures features at different scales utilizing different convolution kernel sizes in its multiple branches. This multibranch design allows GoogLeNet to learn diverse representations and handle complex patterns effectively. Thus, recent works [28] in metric learning usually use it as the backbone network.

The cornerstone of our approach is the multisimilarity (MS) loss [28], which is instrumental in learning a robust embedding space. The MS loss is an outstanding contribution to the field of deep metric learning, particularly in the unsupervised domain. It operates on the principle of general pair weighting (GPW), which provides a unified perspective on the sampling problem by analyzing the gradient contributions of different pairs. This framework is pivotal in understanding and improving upon existing pair-based loss functions.

MS loss involves identifying informative positive and negative pairs, while the weighting step assigns appropriate weights to these pairs based on their relative similarities. It ensures that the loss function fully considers:

1) Relative similarity between positive samples, promoting the grouping of similar patterns.

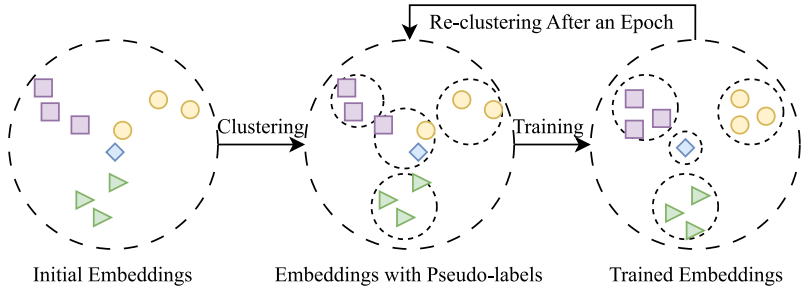

Fig. 6. Illustration of the unsupervised metric learning methodology. The training is guided by the pseudo-labels generated by clustering. The reclustering of data is repeated before each training epoch.

2) Dissimilarity among negative samples, encouraging the separation of distinct patterns.

Incorporating these considerations, MS loss is defined as

$$\mathcal{L}_{\mathrm{MS}}(P,N) = \frac{1}{|P|}\sum_{(i,j)\in P}\log\left[1+\exp\left(-\gamma_P(s_{ij}-m)\right)\right] + \frac{1}{|N|}\sum_{(i,k)\in N}\log\left[1+\exp(-\gamma_N(m-s_{ik}))\right] \quad (8)$$

where the sets P and N consist of positive and negative pairs, respectively. Scaling parameters γ_P and γ_N control the strength of penalties. The margin m separates positive and negative pairs. The similarity scores for positive and negative pairs are denoted by s_{ij} and s_{ik}, respectively.

The MS loss is visualized in Fig. 5. It draws positive pairs closer while pushing negative pairs further apart. During gradient descent, positive pairs with low similarity and negative pairs with high similarity are given higher weights. It ensures that the training process focuses more on the challenging samples, which contributes to the convergence speed and final performance.

To train GoogLeNet with MS loss, an essential step is leveraging KMeans clustering to generate the pseudo-labels that can guide the training process. At the start of each training epoch, GoogLeNet serves as the backbone to extract embeddings from the training data. These embeddings are then clustered using KMeans algorithm, which partitions the data into a predefined number of groups. This clustering step is crucial, as it automatically assigns pseudo-labels: embeddings within the same cluster are treated as belonging to the same class, forming positive pairs, while those in different clusters are considered negative pairs. By performing KMeans clustering iteratively at each epoch, the model continuously refines these pseudo-labels, enabling the GoogLeNet with MS loss to learn effectively from both intracluster similarities and intercluster dissimilarities. This dynamic approach harnesses the power of KMeans to reveal underlying data structures without requiring manual annotations, ultimately enhancing the robustness of the unsupervised metric learning process.

The training of our pattern embedding model is summarized in Algorithm 1. To generate the pseudo-labels for unsupervised learning, we use KMeans clustering to group similar patterns together, as shown in lines 3 and 4. Although the embeddings cannot reflect similarity well at the beginning, the training with (8) (lines 6–9) can adjust the model and learn a better distance metric. In addition to the effectiveness of MS loss, the underlying reason why the training can be successful is that similar patterns are naturally more likely to be brought together through the training process.

Algorithm 1 Pattern Embedding Training

Input: Number of epochs #*Epochs*, number of steps per epoch #*Steps*, number of groups M.
Output: Pattern embedding model $\mathcal{F}$.

```
1: Initialize the DNN model F;
2: for epoch ∈ [1, #Epochs] do
3:     Generate the pattern embeddings for all training data;
4:     Cluster the embeddings into M groups using KMeans;
5:     for step ∈ [1, #Steps] do
6:         Randomly select C groups from the clusters;
7:         Randomly select K samples from each group;
8:         For each sample, compute the loss L_MS(P, N);
9:         Optimize the model via gradient descent;
10:    end for
11: end for
```

Fig. 6 illustrates the unsupervised metric learning methodology. The pattern embedding model maps input patterns to specific points within the embedding space. Before each training epoch, a clustering algorithm groups these embeddings, assigning a unique pseudo-label to each cluster. These pseudo-labels play a crucial role in guiding the training process alongside the loss function. By reclustering the data at each epoch, the pseudo-labels are continuously updated, allowing them to better capture the inherent similarities among patterns. Over the course of training, this iterative process ensures that similar patterns gradually converge within the embedding space, refining the model's ability to learn meaningful representations.

B. Vector Database

A vector database is meticulously crafted to efficiently manage high-dimensional vectors. The vectors are methodically organized to enable swift and accurate retrieval operations, guided by similarity metrics. The underlying architecture of a vector database is tailored to bolster essential capabilities,

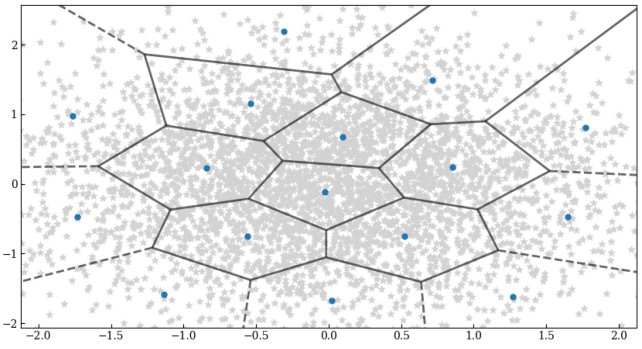

Fig. 7. Toy example of the clustering step in IVF. Clustering allows the database to partition the space and represent each subspace with a centroid. When a query vector is provided, the search algorithm focuses on the nearest subspace, effectively minimizing unnecessary computational overhead.

including retrieval predicated on similarity and seamless integration with other workflows. To ensure robust adaptability, our framework incorporates a powerful inverted file indexing (IVF) method for pattern retrieval.

IVF is a technique used in vector databases to enhance search efficiency by partitioning the vector space into clusters. The basic steps for constructing the IVF index of a vector database are shown as follows.

1) *Clustering:* The vector space is divided into multiple clusters using KMeans clustering algorithm. Each cluster is represented by its centroid.
2) *Indexing:* Each vector in the database is assigned to the nearest centroid. After that, we create an inverted index, which maps each centroid to the vectors within the corresponding cluster.
3) *Search:* Given a query vector, the search algorithm identifies the nearest centroids and only searches within the corresponding clusters, significantly reducing the search space and improving the efficiency.

Fig. 7 presents a toy example of the clustering step in IVF. Clustering enables the database to partition the space and represent each subspace with a centroid. Given a query vector, the search algorithm can focus on the nearest subspace, thereby avoiding redundant computational overhead.

To accelerate the retrieval process, we combine IVF with the product quantization (PQ) strategy. PQ is a method used to compress high-dimensional vectors into lower-dimensional representations, which further enhances the efficiency of vector searches. In PQ, each embedding vector is divided into smaller subvectors. Each subvector is quantized independently by mapping it to the nearest point in a precomputed codebook. This process reduces the dimensionality and storage requirements. Our IVF index is built upon the quantized vectors to reduce the computational burden.

C. Pattern Clustering

Traditional clustering methods like KMeans can be utilized in pattern clustering. However, as the number of points increases, the computation of KMeans becomes more and more time-consuming, and the convergence becomes more and more difficult. This inhibits KMeans or similar algorithms to be applied in a huge pattern database. To solve the problem, we propose an efficient pattern clustering method based on hierarchical K-neighbor graph partitioning. Coarse-grained partitioning is conducted on the graph to harness the superior efficiency of graph partitioning algorithms. Fine-grained clustering is applied to the subgraphs to utilize the high quality of KMeans. The combination of graph partition and KMeans clustering makes is possible to process massive patterns in an acceptable time. Algorithm 2 summarizes our algorithm, whose the major steps are described as follows.

Algorithm 2 Hierarchical K-Neighbor Graph Partitioning

Input: Layout tiles C, pattern embedding model F, target number of clusters N, number of neighbors K, coarse-grained threshold T_C, fine-grained threshold T_F.
Output: Pattern clusters $\mathcal{C}$.

1: Initialize the coarse-grained stack $\mathcal{S} = \emptyset$;
2: Initialize the fine-grained queue $\mathcal{Q} = \emptyset$;
3: Initialize the clusters $\mathcal{C} = \emptyset$;
4: // K-neighbor Graph Construction
5: Obtain the pattern embeddings $E = F(C)$;
6: Construct the K-neighbor graph $\mathcal{G}$ based on E;
7: // Coarse-grained Partitioning
8: Identify all connected components $\mathcal{G}_S$ in the $\mathcal{G}$;
9: Put the components with more than T_C nodes into $\mathcal{S}$, and others into $\mathcal{Q}$;
10: **while** $\mathcal{S}$ is non-empty **do**
11: Pop $\mathcal{G}_i$ from $\mathcal{S}$;
12: Bipartition $\mathcal{G}_i$ into $\mathcal{G}_{i1}$ and $\mathcal{G}_{i2}$;
13: Put the subgraphs with more than T_C nodes into $\mathcal{S}$, and others into $\mathcal{Q}$;
14: **end while**
15: // Fine-grained Clustering
16: **while** $\mathcal{Q}$ is non-empty **do**
17: Dequeue $\mathcal{G}_i$ from $\mathcal{Q}$;
18: **if** intra-cluster discrepancy of $\mathcal{G}_i$ is smaller than T_F **then**
19: Put $\mathcal{G}_i$ into $\mathcal{C}$;
20: **else**
21: Divide $\mathcal{G}_i$ into $\{\mathcal{G}_{i1}, \mathcal{G}_{i2}, \cdots\cdots\}$ using KMeans;
22: Put the clusters with intra-cluster discrepancies larger than T_F into $\mathcal{Q}$, and others into $\mathcal{C}$;
23: **end if**
24: **end while**
25: // Final Adjustment
26: Merge the clusters in $\mathcal{C}$ if $|\mathcal{C}| > N$;
27: Split the clusters in $\mathcal{C}$ if $|\mathcal{C}| < N$;
28: **return** $\mathcal{C}$;

K-Neighbor Graph Construction: The first step in our algorithm is to obtain a vector representation for each layout tile from the pattern embedding model (line 5). Once the embeddings are obtained, we construct a K-neighbor graph (line 6). Each node in the graph represents a layout tile, and each edge connects a node to its K nearest neighbors based on the Euclidean distance in the embedding space. This graph serves as the foundation for our subsequent partition phases.

The graph construction process can be accelerated by our pattern database, since nearest neighbor searching is one of its basic functionality.

Coarse-Grained Partition: In the coarse-grained partition phase, we aim to split the K-neighbor graph into coarse-grained subgraphs. This is achieved by first identifying all connected components in the graph (line 8). For each component, we check the number of nodes it contains (line 9). If the size is larger than a predefined coarse-grained threshold T_C, we consider it for further splitting and place the corresponding subgraph into a coarse-grained stack. Otherwise, we place the subgraph into a fine-grained queue for later processing.

We then iteratively pop a subgraph from the coarse-grained stack and apply graph bipartition [29] to split it into two subgraphs (lines 11 and 12). The goal of this bipartition is to minimize the cut between the two resulting subgraphs, ensuring that they are as disjoint as possible. The bipartition results are then placed back into the coarse-grained stack or fine-grained queue based on their sizes (line 13). This process continues until all subgraphs in the coarse-grained stack have been processed.

Fine-Grained Clustering: During the fine-grained clustering phase, we focus on splitting the smaller subgraphs in the fine-grained queue. For each subgraph $\mathcal{G}_i$, we compute its intracluster discrepancy, which is estimated by randomly sampling data pairs in the cluster and calculating their average Euclidean distance. We put $\mathcal{G}_i$ in the pattern clustering results if its intracluster discrepancy is smaller than the fine-grained threshold T_F (line 19). Otherwise, we apply the KMeans clustering algorithm to split it into smaller clusters (line 21). The number of clusters is determined by $\max(2, \lfloor \log_{10}(|\mathcal{G}_i|) \rfloor)$. This ensures that large clusters are split into a more balanced number of small clusters. If the intracluster discrepancies of any resulting clusters are still larger than the fine-grained threshold T_F, we place them back into the fine-grained queue for further processing (line 22). This iterative process continues until the fine-grained queue have been processed.

Final Adjustment: After completing both coarse-grained and fine-grained phases, we may end up with a set of subgraphs that does not exactly match the target number of clusters N specified for the final output. To adjust for this, we apply the following rules.

1) If the number of subgraphs is larger than the target number N, we merge them by clustering their centroids using KMeans. The number of clusters in this final KMeans step is set to N (line 26).
2) If the number of subgraphs is smaller than the target number N, we continuously split the largest subgraph using the same strategy in the fine-grained clustering phase until we have N subgraphs (line 27).

In our algorithm, four key parameters influence clustering performance: the number of neighbors K, the coarse-grained threshold T_C, the fine-grained threshold T_F, and the target number of clusters N. Among these, T_F is the most critical, as it controls intracluster discrepancies—smaller values enforce stricter similarity constraints, leading to a larger number of fine-grained clusters. Based on empirical evaluation, we set $T_F = 0.5$. The number of neighbors K affects both clustering quality and efficiency, where a larger K improves similarity-based grouping but increases runtime; we set $K = 16$ for a balanced tradeoff. The coarse-grained threshold T_C determines the balance between cluster uniformity and intracluster discrepancies. We empirically set it to $1/100$ of the total number of data points. Lastly, N is primarily introduced for comparison with KMeans clustering, and we consider that T_F plays a more significant role in determining clustering quality. The final adjustment step can be skipped if the application does not require a specific number of clusters.

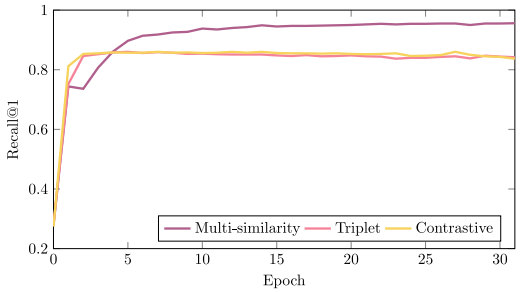

Fig. 8. Recall@1 curves of different loss functions: (1) MS, (2) triplet, (3) contrastive.

D. Pattern Retrieval for HSD

Existing methods for HSD often rely on complex simulations, heuristic rules, or DNN-based classification models, which can be computationally expensive and difficult to generalize. To address these challenges, we propose an algorithm that uses pattern retrieval for efficient and accurate HSD. The algorithm leverages a pattern database constructed from training layout tiles, where each pattern embedding in the database is associated with a label indicating whether it is a hotspot or not. Given a test sample, the algorithm retrieves similar patterns from the database and classifies the test sample as a hotspot if the ratio of retrieved hotspots exceeds a predefined threshold T_H.

The algorithm's reliance on an extensible pattern database is a significant strength. As new data is generated in the semiconductor manufacturing process, it can be easily incorporated into the database. This ensures that the HSD method remains up-to-date and can handle a wide variety of patterns and features. By using pattern embeddings and similarity metrics, the algorithm can efficiently retrieve similar patterns from the database. This allows for rapid classification of test samples, making the algorithm scalable for large-scale scenarios.

E. ILT Acceleration via Solution Reusing

The acceleration of ILT is a significant advancement in computational lithography, aimed at reducing the computational burden associated with mask optimization. Our approach builds upon the concept introduced by AdaOPC [30], enhancing it with finetuning to achieve superior results. To intialize the database for ILT, we obtain the optimized masks for previously encountered patterns using the ILT algorithm introduced in Section II-A. At each ILT iteration, we generate the mask with (3), predict the printed image through (2), compute the loss function by (4), and carry out optimization

TABLE I
OVERALL COMPARISON BETWEEN OUR FRAMEWORK AND THE BASELINES

Task	Metric	Baseline	Ours	Note
Pattern Embedding	Recall@1	0.860	**0.956**	The baseline is the triplet loss used in [20]
Vector Database	Query Time	28.4s	**10.0s**	1M points. The baseline is HNSW used in [30]
Pattern Clustering	Runtime	15827s	**2607s**	1M points. The baseline is KMeans clustering.
Retrieval-based HSD	F1 Score	0.818	**0.968**	The baseline is ResNet18 used in [31].
ILT Solution Reusing	L2 Loss	18387	**16423**	The baseline is the default reusing scheme in [30]
ILT Parameter Optimization	PVB	17306	**16272**	The baseline is KMeans clustering.
Source Optimization	L2 Loss	16657	**16628**	The baseline is KMeans clustering.

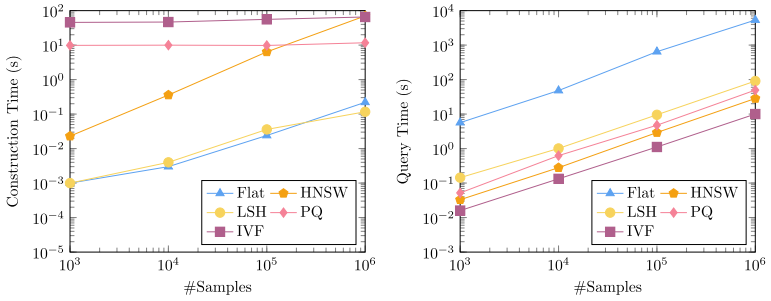

Fig. 9. Runtime comparison between different indexing methods in terms of (a) construction time and (b) query time.

using (5). The initialized database acts as a knowledge base, storing the solutions to previously encountered patterns. When new patterns are introduced to the system, the first step is to query this database for similar patterns. Upon identifying similar patterns in the database, we employ a reuse-and-finetune strategy:

1) *Solution Reusing:* We retrieve the optimized masks corresponding to the similar patterns and use them as starting points for the optimization of new patterns. Note that we consider two embeddings to be similar if the distance between them is smaller than a threshold. If a similar pattern is found, we reuse its result to reduce the runtime required for mask optimization while maintaining high-quality outcomes.
2) *Finetuning:* Unlike AdaOPC, which stops at reusing existing solutions by default, our method incorporates an additional finetuning phase. This involves running a reduced number of optimization iterations to refine the reused mask, tailoring it to the specificities of the new pattern.

F. ILT/Source Parameter Optimization

Optimizing the parameters for ILT and source configuration is crucial for achieving high-quality lithographic results. Our framework employs optimization methods to finetune these parameters on representative patterns.

ILT Parameter Optimization With TPE: For ILT parameter optimization, we utilize the tree-structured Parzen estimator (TPE) [32], a Bayesian optimization method. TPE is an efficient approach that models the probability distribution of the parameters and selects new parameter values by maximizing an acquisition function based on the historical performance of the parameters.

TABLE II
COMPARISON AMONG METRIC LEARNING METHODS

Method	Recall@1	Inference Time
ResNet + MS	0.932	1.4 ms/sample
AttenNet + MS	0.953	1.8 ms/sample
GoogLeNet + Triplet	0.860	**1.2** ms/sample
GoogLeNet + Contrastive	0.837	**1.2** ms/sample
GoogLeNet + MS	**0.956**	**1.2** ms/sample

The TPE method works as follows.

1) *Modeling:* Construct a probabilistic model of the objective function based on past evaluations. Specifically, the Parzen estimator method is utilized to fit the mixture-of-Gaussian model, which is efficient and effective.
2) *Acquisition:* Use the model to choose the next set of parameters to evaluate by maximizing the expected improvement (EI) over the best observed performance. The EI function considers the expected value and the uncertainty simultaneously to strike a balance between exploitation and exploration.
3) *Evaluation:* Apply the selected parameters to the representative patterns and observe the results.
4) *Update:* Incorporate the new results into the model and repeat the process to refine the ILT performance.

This method allows us to systematically explore the parameter space and converge toward the optimal settings for ILT. The search space of ILT parameter optimization is determined by the steepness θ_M and the offset γ_M in (3), the scales α and β in (4), and the step size in the gradient descent optimizer. By exploring this search space, we can refine the ILT algorithm introduced in Section II-A in an automatic manner.

Source Optimization With Gradient Descent: For source optimization, we employ the Abbe model in (1) and utilize gradient descent to optimize the source matrix S. Gradient

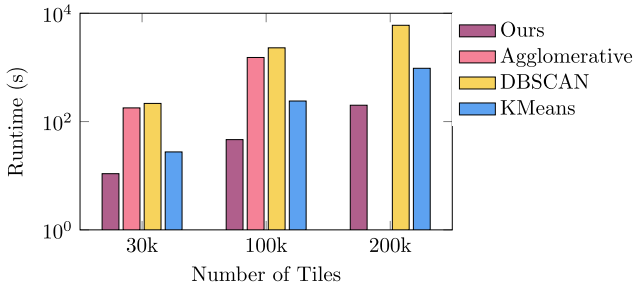

Fig. 10. Runtime comparison among different pattern clustering algorithms. The runtime of agglomerative clustering with 200k tiles is unavailable due to insufficient memory. In all tested cases, our method and KMeans outperform other approaches, demonstrating superior scalability for large-scale applications. Moreover, our approach is consistently faster than KMeans in all tested cases.

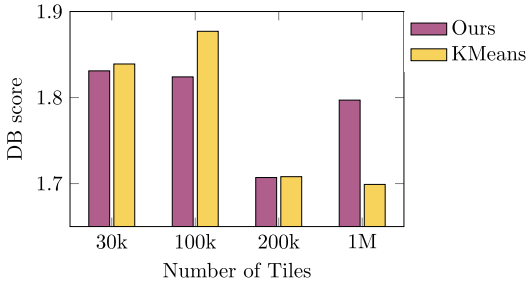

Fig. 11. Clustering quality comparison between KMeans and our method. When the number of data points is small, KMeans performs better than our approach. However, when the amount of data is huge, our method outperforms KMeans, making it suitable for large-scale datasets. Note that our approach performs better with 1M tiles than with 200K tiles because its parameters are primarily optimized for the 1M case.

descent iteratively adjusts the source parameters in the direction opposite to the gradient of the objective function, which in this case, is the L2 distance between the printed and target patterns. This strategy ensures that we explore the parameter space effectively and make informed adjustments, leading to improved lithographic accuracy and efficiency.

TABLE III
COMPARISON AMONG PATTERN CLUSTERING METHODS

Method	#Tiles	VRC score	DB score	Runtime
KMeans	1M	6709	1.7	15827 s
Ours	1M	**7203**	**1.8**	**2607 s**
Improvement	-	**7.4%**	**5.9%**	**6×**

IV. EXPERIMENTS

This section begins by outlining the experimental settings. Subsequently, we assess the proposed methodologies with respect to pattern embedding, vector database construction, and pattern clustering. Following this, we deploy our pattern database across various downstream applications, including retrieval-based HSD, ILT solution reusing, and ILT/source parameter optimization. As illustrated by Table I, our framework outperforms the baseline models across all six tasks. The subsequent content in this section will delve into the experimental results in greater detail.

A. Experimental Settings

We implement all DNN models with PyTorch [33] and test them on RTX Titan GPU. We develop ILT-related applications based on the OpenILT framework [25]. For HSD, we check the EPE after ILT, and a layout tile with EPE is considered as a hotspot. The pattern retrieval is accelerated using Faiss [26]. Inspired by [30] and [34], we obtain data by cropping the first metal layer of the *gcd* and *aes* layout generated by OpenROAD [35], which contains diverse shapes in different sizes. Each tile has a size of 2048×2048 nm^2 and is rasterized to have 256×256 pixels.

B. Pattern Embedding

In this experiment, we compare our embedding model with related works to show that our method can better extract representative features from layout patterns. We train the embedding model in an unsupervised metric learning setting with an embedding size of 256. Around 200 k patterns from the *aes* layout are used for training. The comparison involves three models: 1) the ResNet [36] used in [31]; 2) the AttenNet designed by [37]; and 3) the GoogLeNet we use. The triplet loss used in [20], the contrastive loss used in [30], and our MS loss are also compared in our experiment.

To compare the methods, we use the recall@1 metric. In a retrieval scenario, if the top-ranked result retrieved by the model matches the ground truth positive example, it contributes to the recall@1 score. Table II provides a comprehensive comparison of various metric learning configurations. These results indicate that our method outperforms others, achieving the highest recall@1 score while maintaining the lowest latency. Among the configurations using the MS loss, the GoogLeNet architecture delivers the best recall@1 performance. Among the configurations utilizing GoogLeNet, the MS loss yields the highest recall@1 score. As shown in Fig. 8, although triplet and contrastive losses converge quickly, our method based on MS loss achieves remarkably higher recall@1 after training. These results suggest that our approach can effectively learn the underlying pattern features.

C. Vector Database

In this experiment, we evaluate the performance of different indexing methods in terms of runtime and recall, demonstrating that our IVF algorithm achieves better efficiency while maintaining high pattern retrieval quality. The following methods [26] are adopted as our baselines.

1) *Flat:* It performs a brute-force search by computing the distance between the query vector and all the vectors in the database. It is accurate but slow in large datasets.
2) *Hierarchical Navigable Small World (HNSW):* HNSW is a graph-based method for approximate nearest neighbor search. It builds a layered graph where each layer is a navigable small world graph. Searches start at the top layer and traverse down, which allows for very efficient search times. It is adopted in AdaOPC [30].
3) *Locality-Sensitive Hashing (LSH):* It is an approximate search method that hashes input items so that similar items map to the same "buckets" with high probability.

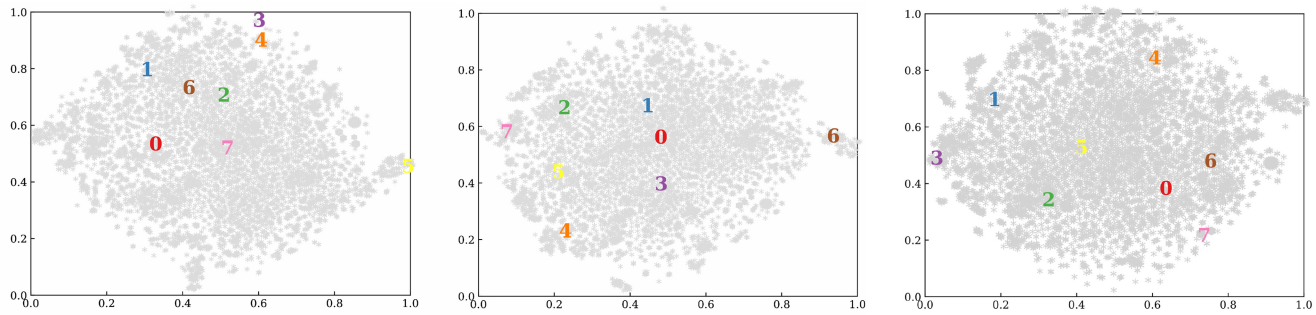

Fig. 12. Representative pattern selection results from (a) random selection, (b) KMeans clustering, and (c) our method.

TABLE IV
COMPARISON AMONG HSD METHODS

Testcase	Method	Precision	Recall	F1 Score
gcd (3.5k tiles)	ResNet [31]	0.942	0.943	0.943
	AttenNet [20]	**0.948**	0.899	0.923
	Ours	0.931	**0.962**	**0.946**
aes (200k tiles)	ResNet	0.957	0.712	0.816
	AttenNet	0.945	0.741	0.831
	Ours	**0.960**	**0.977**	**0.968**
Average (per tile)	ResNet	0.957	0.716	0.818
	AttenNet	0.945	0.744	0.833
	Ours	**0.960**	**0.977**	**0.968**

It is faster than brute-force search for large datasets but trades off accuracy for speed.

4) *PQ:* It compresses vectors by dividing them into smaller subvectors and quantizing each subvector into one of several points. During search, distances are approximated using these points. It is a balance between memory usage and accuracy.

Since the pattern database is built for querying similar patterns, the query time is a critical concern. As shown in Fig. 9, our IVF method achieves the lowest query time, which is 2× faster than the HNSW method used in [30]. Additionally, we compare IVF and HNSW on the retrieval recall, which is the ratio of retrieved samples that match the Flat method's results. IVF has a recall of 100%, while HNSW has a poorer recall of 97.5%. Thus, IVF can strongly boost the performance of our pattern database.

D. Pattern Clustering

In this experiment, we compare various pattern clustering methods to demonstrate the superior efficiency of our proposed approach. We use layout tiles randomly cropped from the gcd and aes layouts, and use different pattern clustering methods to divide them into one thousand groups. To compare pattern clustering methods, we adopt the variance ratio criterion (VRC) [38] and Davies-Bouldin (DB) [39] scores as quality metrics. The VRC score is defined as the ratio of between-cluster dispersion to within-cluster dispersion. The DB score measures the average similarity of each cluster to its most similar cluster, where similarity is the ratio of within-cluster distances to between-cluster distances. Clusters that are more distinct and less dispersed yield better scores. To evaluate the methods' efficiency, their runtimes are also measured in this experiment.

We use several algorithms that have been used for pattern clustering as our baselines, including: 1) KMeans [40]; 2) DBSCAN [41]; and 3) agglomerative clustering [42]. Fig. 10 presents a runtime comparison of these pattern clustering algorithms. Notably, the runtime for agglomerative clustering with 200 K tiles is unavailable due to memory constraints. Across all tested scenarios, our proposed method, along with KMeans clustering, consistently outperforms other approaches. This superior performance highlights their scalability, making them well-suited for large-scale applications. Therefore, we select our method and KMeans for the experiment with more tiles.

As summarized in Table III, with 1 million tiles, the proposed method not only runs much faster than KMeans, but also achieves better clustering quality, as indicated by the VRC and DB scores. Given the large scale data collection, our method can achieve 6× speedup with 7.4% enhancement on VRC score and 5.9% improvement on DB score. Notably, as shown in Fig. 11, KMeans performs better than our approach when the dataset is small. However, as the dataset size increases, our method surpasses KMeans in performance, making it more suitable for large-scale datasets. When the number of data is small, users may switch to KMeans for better clustering quality, or finetune the parameters of our approach to improve the result.

The cluster centroids can be viewed as representative patterns within the complete dataset. These representative patterns can be utilized for ILT/source parameter optimization. In this experiment, we visually demonstrate the effectiveness of our approach in selecting representative patterns. Fig. 12 presents a toy example comparing the selection results obtained from random sampling, KMeans, and our method. The proposed method can get representative patterns that are evenly distributed in the space, highlighting its suitability for representative pattern selection.

E. Retrieval-Based HSD

In this experiment, we employ the proposed pattern retrieval approach in our retrieval-based HSD flow, demonstrating that our pattern database enables better HSD accuracy. To

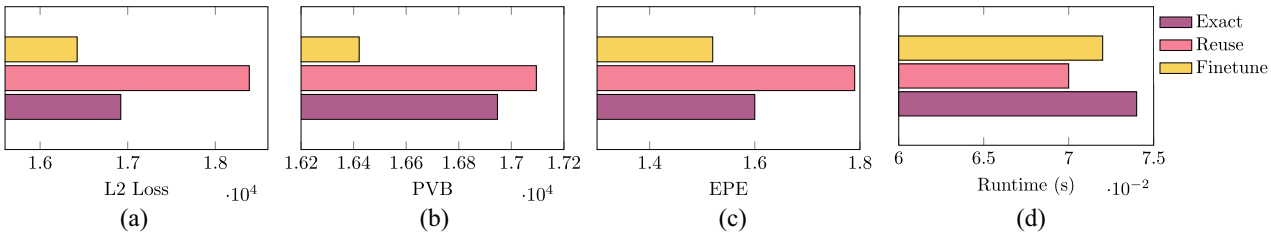

Fig. 13. Comparison on ILT acceleration via solution reusing, in terms of (a) L2, (b) PVB, (c) EPE, and (d) runtime.

show the effectiveness of retrieval-based HSD method, we compare it with state-of-the-art HSD methods, which use the ResNet18 model [31] and an attention-based model [37]. We test these methods on both *gcd* and *aes* layouts. The HSD labels are generated by simulating the optimized mask using the Hopkins model (2) and measuring the EPE violations. A layout tile containing EPE violations is considered to be a hotspot. Precision, recall, and F1 score are adopted as the metrics for HSD. *Precision* corresponds to the proportion of data that our model correctly identifies as hotspots out of all the data it predicts as hotspots, which is defined as

$$p = \frac{|Pred \cap Truth|}{|Pred|} \tag{9}$$

where *Pred* contains the predicted hotspots and *Truth* includes the ground truth. *Recall* measures how many of the hotspots that should be detected are actually identified by our model, which is defined as

$$r = \frac{|Pred \cap Truth|}{|Truth|}. \tag{10}$$

To have a unified metric, we also use the *F1 score* to evaluate the performance of different methods. It is defined as the harmonic mean of precision p and recall r, which comprehensively measures the model's exactness and correctness. The F1 score is calculated by the following equation:

$$F1 = \frac{2pr}{p + r}. \tag{11}$$

As presented by Table IV, our method achieves significantly better performance than existing methods, especially on recall and F1 score. On average, our method improves the recall and F1 score by 0.143 and 0.077, respectively. These results indicate that our method can better avoid missing hotspots. Given retrieval-based HSD's extensible feature and superior performance, it can become a promising candidate for traditional classification-based approaches.

F. ILT Acceleration via Solution Reusing

In this experiment, the proposed pattern retrieval approach is utilized to reuse existing results and accelerate the ILT for 10k masks. OpenILT [25] is employed for mask optimization. We run ILT for 20 iterations by default and five iterations for finetuning. We compare our solution reusing scheme with the default scheme from [30], which reuses the results from similar patterns without finetuning. We denote the scheme from [30] with "Reuse," and ours with "Finetune." In addition, we also implement an "Exact" method, which only reuses patterns that can be exactly matched pixel-by-pixel in the optimized masks.

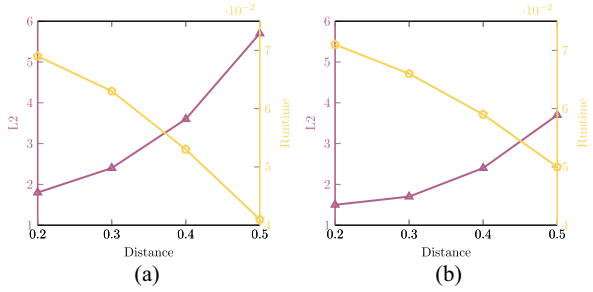

Fig. 14. Comparison on ILT acceleration via solution reusing with varying distance threshold. The tested methods are (a) reuse and (b) finetune.

Fig. 13 compares the three schemes in terms of L2, PVB, EPE, and runtime. Our method achieves the best L2, PVB, and EPE, which indicates that our finetuning can not only save runtime by reducing optimization iterations but also improve the quality of results by exploiting the similarity between patterns.

In Fig. 14, we evaluate the L2 loss and runtime of these methods by changing their reuse threshold. Although the "Reuse" scheme is slightly faster, its quality falls behind our method, which can achieve a better tradeoff between efficiency and effectiveness.

G. ILT/Source Parameter Optimization

In this experiment, we demonstrate that our pattern clustering approach efficiently identifies representative patterns in large-scale datasets, achieving faster processing and higher quality, ultimately enhancing ILT/source parameter optimization. To optimize ILT/source parameters, we select 200 data from 200k layout tiles using the proposed representative pattern selection method and test the optimized parameters using 2k data. We run TPE for 50 iterations to optimize the ILT parameters. For source optimization, we begin with a circular source and run gradient descent for 16 iterations. We compare three representative pattern selection methods: random sampling, KMeans clustering, and our method. As shown in Fig. 15(a), (b), and (c), our method can achieve a more balanced performance than random sampling and KMeans in ILT parameter optimization. In terms of source optimization, our method performs better than KMeans and random sampling. Since our method is significantly faster than KMeans, it is more promising in accelerating ILT/source parameter optimization.

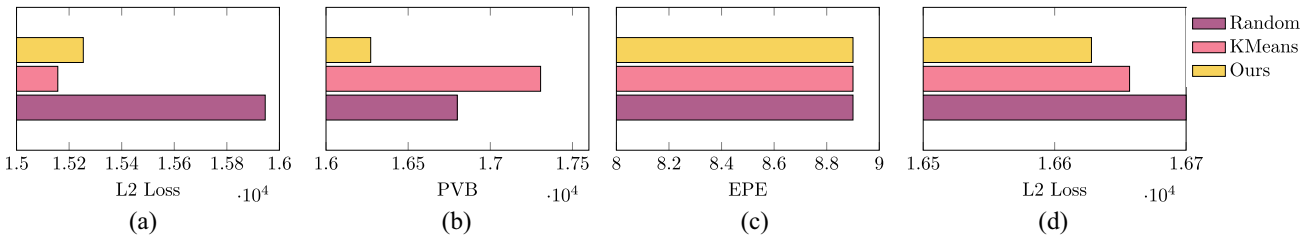

Fig. 15. Comparison on ILT/source parameter optimization. The ILT parameter optimization results are compared in terms of (a) L2, (b) PVB, and (c) EPE, and (d) compares the L2 of source optimization.

V. Conclusion

In conclusion, the development of our pattern database for computational lithography represents a significant leap forward in the pursuit of more efficient and accurate computational lithography. The framework introduced in this article, encompassing an unsupervised metric learning method for pattern embedding, a vector database for pattern retrieval, and a hierarchical K-neighbor graph partition method for pattern clustering, has demonstrated its potential to revolutionize the field. The application of these methods to enhance lithography HSD via pattern retrieval has highlighted the improvement in recall and F1 score. The ILT strategy of solution reusing has underscored the benefits of leveraging past knowledge to reduce computational demands. Furthermore, the hierarchical K-neighbor graph partition method has provided a systematic approach for pattern clustering and the exploration of ILT/source parameters. The potential of pattern database for reduced turnaround times, lower costs, and improved yields is immense, making substantial progress in computational lithography. Our code is available at https://github.com/shelljane/litho-pattbase.

References

[1] J.-S. Park et al., "An efficient rule-based OPC approach using a DRC tool for 0.18/spl mu/m ASIC," in *Proc. IEEE Int. Symp. Qual. Electron. Design (ISQED)*, 2000, pp. 81–85.

[2] S. Banerjee, K. B. Agarwal, and M. Orshansky, "SMATO: Simultaneous mask and target optimization for improving lithographic process window," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, 2010, pp. 100–106.

[3] J. Kuang, W.-K. Chow, and E. F. Young, "A robust approach for process variation aware mask optimization," in *Proc. IEEE/ACM Proc. Design, Autom. Test Eurpoe (DATE)*, 2015, pp. 1591–1594.

[4] J.-R. Gao, X. Xu, B. Yu, and D. Z. Pan, "MOSAIC: Mask optimizing solution with process window aware inverse correction," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, 2014, pp. 1–6.

[5] Z. Yu, G. Chen, Y. Ma, and B. Yu, "A GPU-enabled level-set method for mask optimization," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 42, no. 2, pp. 594–605, Feb. 2023.

[6] S. Yin et al., "FuILT: Full chip ILT system with boundary healing," in *Proc. ACM Int. Symp. Phys. Design (ISPD)*, 2024, pp. 13–20.

[7] S. Sun, F. Yang, B. Yu, L. Shang, and X. Zeng, "Efficient ILT via multi-level lithography simulation," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, 2023, pp. 1–6.

[8] S. Sun, F. Yang, B. Yu, L. Shang, D. Zhou, and X. Zeng, "Efficient ILT via multigrid-schwartz method," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, 2024, p. 195.

[9] H. Yang, S. Li, Z. Deng, Y. Ma, B. Yu, and E. F. Young, "GAN-OPC: Mask optimization with lithography-guided generative adversarial nets," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 10, pp. 2822–2834, Oct. 2020.

[10] B. Jiang, L. Liu, Y. Ma, B. Yu, and E. F. Young, "Neural-ILT 2.0: Migrating ILT to domain-specific and multitask-enabled neural network," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 41, no. 8, pp. 2671–2684, Aug. 2022.

[11] G. Chen, Z. Yu, H. Liu, Y. Ma, and B. Yu, "DevelSet: Deep neural level set for instant mask optimization," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, 2021, pp. 1–9.

[12] G. Chen, W. Chen, Q. Sun, Y. Ma, H. Yang, and B. Yu, "DAMO: Deep agile mask optimization for full-chip scale," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 41, no. 9, pp. 3118–3131, Sep. 2022.

[13] X. Liang, Y. Ouyang, H. Yang, B. Yu, and Y. Ma, "RL-OPC: Mask optimization with deep reinforcement learning," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst. (TCAD)*, vol. 43, no. 1, pp. 340–351, Jan. 2024.

[14] X. Liang, H. Yang, K. Liu, B. Yu, and Y. Ma, "CAMO: Correlation-aware mask optimization with modulated reinforcement learning," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, 2024, pp. 1–6.

[15] H. Yao, S. Sinha, C. Chiang, X. Hong, and Y. Cai, "Efficient process-hotspot detection using range pattern matching," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, 2006, pp. 625–632.

[16] S.-Y. Lin, J.-Y. Chen, J.-C. Li, W.-Y. Wen, and S.-C. Chang, "A novel fuzzy matching model for lithography hotspot detection," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, 2013, p. 68.

[17] J. P. Cain, M. Fakhry, P. Pathak, J. Sweis, F. E. Gennari, and Y.-C. Lai, "Pattern-based analytics to estimate and track yield risk of designs down to 7nm," in *Proc. SPIE Adv. Lithography*, 2017, pp. 1–21.

[18] Y. Jiang, F. Yang, B. Yu, D. Zhou, and X. Zeng, "Efficient layout hotspot detection via binarized residual neural network ensemble," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 40, no. 7, pp. 1476–1488, Jul. 2021.

[19] H. Yang, S. Li, C. Tabery, B. Lin, and B. Yu, "Bridging the gap between layout pattern sampling and hotspot detection via batch active learning," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 40, no. 7, pp. 1464–1475, Jul. 2021.

[20] H. Geng, H. Yang, L. Zhang, F. Yang, X. Zeng, and B. Yu, "Hotspot detection via attention-based deep layout metric learning," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 41, no. 8, pp. 2685–2698, Aug. 2022.

[21] R. Chen, W. Zhong, H. Yang, H. Geng, X. Zeng, and B. Yu, "Faster region-based hotspot detection," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, 2019, p. 146.

[22] B. Zhu et al., "Hotspot detection via multi-task learning and transformer encoder," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, 2021, pp. 1–8.

[23] W. Zhao et al., "AdaOPC: A self-adaptive mask optimization framework for real design patterns," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, to be published.

[24] G. Chen, Z. Wang, B. Yu, D. Z. Pan, and M. D. F. Wong, "Ultrafast source mask optimization via conditional discrete diffusion," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 43, no. 7, pp. 2140–2150, Jul. 2024.

[25] S. Zheng, B. Yu, and M. Wong, "OpenILT: An open source inverse lithography technique framework," in *Proc. IEEE Int. Conf. ASIC (ASICON)*, 2023, pp. 1–4.

[26] M. Douze et al., "The Faiss library," 2024, *arXiv:2401.08281*.

[27] C. Szegedy et al., "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2015, pp. 1–12.

[28] X. Wang, X. Han, W. Huang, D. Dong, and M. R. Scott, "Multi-similarity loss with general pair weighting for deep metric learning," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2019, pp. 5022–5030.

[29] G. Karypis and V. Kumar, "METIS: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices," Dept. Comput. Sci. Eng., Univ. Minnesota, Minneapolis, MN, USA, Rep. 97-061, 1997.

[30] W. Zhao et al., "AdaOPC: A self-adaptive mask optimization framework for real design patterns," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, 2022, p. 123.
[31] Y. Jiang, F. Yang, H. Zhu, B. Yu, D. Zhou, and X. Zeng, "Efficient layout hotspot detection via binarized residual neural network," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, 2019, p. 147.
[32] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," in *Proc. Annu. Conf. Neural Inf. Process. Syst. (NeurIPS)*, vol. 24, 2011, pp. 1–9.
[33] A. Paszke et al., "Pytorch: An imperative style, high-performance deep learning library," in *Proc. Annu. Conf. Neural Inf. Process. Syst. (NeurIPS)*, vol. 32, 2019, pp. 8026–8037.
[34] S. Zheng, H. Yang, B. Zhu, B. Yu, and M. Wong, "LithoBench: Benchmarking AI computational lithography for semiconductor manufacturing," in *Proc. Annu. Conf. Neural Inf. Process. Syst. (NeurIPS)*, 2023, pp. 1–12.
[35] T. Ajayi et al., "Toward an open-source digital flow: First learnings from the OpenROAD project," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, 2019, p. 76.
[36] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2016, pp. 770–778.
[37] H. Geng et al., "Hotspot detection via attention-based deep layout metric learning," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, 2020, pp. 1–8.
[38] T. Caliński and J. Harabasz, "A dendrite method for cluster analysis," *Commun. Stat. Theory Methods*, vol. 3, no. 1, pp. 1–27, 1974.
[39] D. L. Davies and D. W. Bouldin, "A cluster separation measure," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. PAMI-1, no. 2, pp. 224–227, Apr. 1979.
[40] B. Yu, J.-R. Gao, D. Ding, X. Zeng, and D. Z. Pan, "Accurate lithography hotspot detection based on principal component analysis-support vector machine classifier with hierarchical data clustering," *J. Micro/Nanolithography, MEMS, MOEMS*, vol. 14, no. 1, 2015, Art. no. 11003.
[41] G. Cho, Y. Kwon, P. Kareem, and Y. Shin, "Integrated test pattern extraction and generation for accurate lithography modeling," *IEEE Trans. Semicond. Manuf.*, vol. 35, no. 3, pp. 495–503, Aug. 2022.
[42] Y. Feng et al., "Layout pattern analysis and coverage evaluation in computational lithography," *Opt. Exp.*, vol. 31, no. 5, pp. 8897–8913, 2023.

Su Zheng received the B.Eng. and M.S. degrees from Fudan University, Shanghai, China, in 2019 and 2022, respectively. He is currently pursuing the Ph.D. degree with the Department of Computer Science and Engineering, the Chinese University of Hong Kong, Hong Kong, under the supervision of Prof. B. Yu and Prof. M. D. F. Wong.

His research interest is to solve critical problems in electronic design automation with advanced artificial intelligence methods.

Wenqian Zhao received the B.Sc. and Ph.D. degrees from the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong, in 2019 and 2024, respectively.

His research interests include machine learning for VLSI design automation and hardware-aware deep-learning acceleration.

Shuyuan Sun received the B.E. degree in microelectronics science and engineering from Xidian University, Xi'an, China, in 2020. She is currently pursuing the Ph.D. degree with the State Key Laboratory of Integrated Chips and System, Microelectronics Department, Fudan University, Shanghai, China.

Her current research interests include hotspot detection and optical proximity correction.

Fan Yang (Member, IEEE) received the B.S. degree from Xi'an Jiaotong University, Xi'an, China, in 2003, and the Ph.D. degree from Fudan University, Shanghai, China, in 2008.

He is currently a Full Professor with the Microelectronics Department, Fudan University. His research interests include model order reduction, circuit simulation, high-level synthesis, acceleration of artificial neural networks, yield analysis, and design for manufacturability.

Bei Yu (Senior Member, IEEE) received the Ph.D. degree from The University of Texas at Austin, Austin, TX, USA, in 2014.

He is currently an Associate Professor with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong.

Dr. Yu received eleven Best Paper Awards from top conferences/journals, such as ICCAD 2024, 2021, and 2013, IEEE TSM 2022, DATE 2022, ASPDAC 2021 and 2012, ISPD 2017, and many other awards, including DAC Under-40 Innovator Award, IEEE CEDA Ernest S. Kuh Early Career Award, and Hong Kong RGC Research Fellowship Scheme Award.

Martin D. F. Wong (Life Fellow, IEEE) received the B.Sc. degree in mathematics from the University of Toronto, Toronto, ON, Canada, in 1979, and the M.S. degree in mathematics and the Ph.D. degree in computer science from the University of Illinois at Urbana-Champaign (UIUC), Champaign, IL, USA, 1981 and 1987, respectively.

He was Bruton Centennial Professor of Computer Science with The University of Texas at Austin, Austin, TX, USA, and Edward C. Jordan Professor of ECE with UIUC. From August 2012 to December 2018, he was the Executive Associate Dean of the College of Engineering with UIUC. From January 2019 to August 2023, he was the Dean of Engineering and Choh-Ming Li Professor of Computer Science and Engineering with The Chinese University of Hong Kong, Hong Kong. Since August 2023, he joined Hong Kong Baptist University (HKBU), Hong Kong, as the Provost of HKBU and a Chair Professor of Computer Science. He has published around 500 papers and graduated over 50 Ph.D. students in electronic design automation (EDA). His main research interest is in EDA.

Prof. Wong is a Fellow of ACM.