Learning-Driven Physical Verification

ZHU, Binwu

A Thesis Submitted in Partial Fulfillment
of the Requirements for the Degree of
Doctor of Philosophy
in
Computer Science and Engineering

The Chinese University of Hong Kong
June 2024

Thesis Assessment Committee

Professor WANG Man Ho (Chair)

Professor YU Bei (Thesis Supervisor)

Professor WONG Martin Ding Fat (Thesis Co-Supervisor)

Professor YANG Ming Chang (Committee Member)

Professor YANG Fan (External Examiner)

Abstract

During the chip design flow, the physical verification is undoubtedly a critical step, where the design of an integrated circuit (IC) layout is verified to ensure correct logical functionality and manufacturability. However, with the continuous scaling-down of circuit feature size, the physical verification suffers from severe runtime overhead issues. Recent years have witnessed the great success of machine learning in the electronic design automation (EDA) community. To improve efficiency and guarantee accuracy, this thesis explores artificial intelligence (AI) solutions for a series of important problems in physical verification, including layout hotspot detection, mask optimization, and design rule checking.

In chip manufacturing, design rule checking (DRC) is a critical step in physical verification. DRC requires formatted scripts as the input to design rule checkers. However, these scripts are manually generated in the foundry, which is tedious and error-prone for the generation of thousands of rules in advanced technology nodes. To mitigate this issue, we propose the first DRC script generation framework, leveraging a deep learning-based key information extractor to automatically identify essential arguments from rules and a script translator to organize the extracted arguments into executable DRC scripts.

Printing errors are very common during the lithography process due to the diffraction effect. To reduce printing errors and improve the yield of integrated

circuits, we present L2O-ILT, a deep learning-based model that achieves mask optimization acceleration and keeps remarkable printability performance. Our model structure is implemented by unrolling our customized inverse lithography techniques (ILT) algorithm, and thus the model structure is highly incorporated into prior knowledge of mask optimization. Such an ILT algorithm-inspired model is able to generate an initial mask solution with better performance than previous developing methods, and the high-quality initial mask can be instantly refined to obtain the final solution.

Although mask optimization effectively improves the mask quality, there may still exist hotspots that can potentially lead to open or short-circuit failures, which are caused by lithography variations. To further ensure the manufacturing yield, an accurate hotspot detector is indispensable during the physical verification phase. We propose an end-to-end one-stage hotspot detection framework. We take advantage of the corner and center representation to improve both classification and localization accuracy. Besides, a feature aggregation module is designed to capture global layout information by bridging the relationships between features at different positions.

We further observe that both hotspot detection and OPC require knowledge of layout structure information and they are closely related to the lithography process during chip manufacturing. Based on such strong relationships, we propose that integrating OPC and hotspot detection into a unified deep-learning model will contribute to the performance of both tasks. To bridge the relationship between OPC and hotspot detection, we first pre-train a layout understanding model (LUM) built on the mask modeling technique, which effectively captures the layout geometric information, and then the pre-trained model can be easily fine-tuned on hotspot detection and OPC with limited data. To fully pre-train the layout understanding model, we create a large layout dataset using layout generation techniques, solving

the data-hungry issues.

Based on experimental results, we demonstrate that our proposed deep learning methodologies alleviate critical runtime issues arising in the physical verification. With the development of manufacturing techniques for semiconductors, layouts are becoming more and more complex. We hope that the works presented in this thesis can provide a more robust and effective solution for advanced research in design for manufacturability.

摘要

在芯片設計流程中，物理驗證無疑是一個關鍵的步驟，該步驟驗證了集成電路（IC）佈局的設計，以確保正確的邏輯功能以及可製造性。然而，隨著電路特徵尺寸的不斷縮小，物理驗證面臨嚴重的運行時間開銷問題。近年來，機器學習在電子設計自動化（EDA）的應用中中取得了成功。為了提高效率並確保準確性，本論文探討了物理驗證中一系列重要問題的人工智能（AI）解決方案，包括佈局熱點檢測、掩膜優化和設計規則檢查。

在芯片製造中，設計規則檢查（DRC）是物理驗證中的一個關鍵步驟。DRC要求以格式化的腳本作為設計規則檢查器的輸入。然而，這些腳本在晶圓廠是手動生成的，在先進技術節點中生成數千條規則繁瑣且容易出錯。為了緩解這個問題，我們提出了第一個DRC腳本生成框架，利用基於深度學習的關鍵信息提取器自動識別規則中的基本參數，並設計了腳本轉換器將提取的參數組織成可執行的DRC腳本。

由於衍射效應，印刷錯誤在光刻過程中非常常見。為了減少印刷錯誤並提高集成電路的良率，我們提出了L2O-ILT，一種基於深度學習的模型，實現了掩膜優化加速並保證了良好的印刷性能。我們的模型結構通過展開定制的ILT算法來實現，因此模型結構高度融入了掩膜優化的先前知識。這種受ILT算法啓發的模型能夠生成比以前的開發方法更好性能的初始掩膜解決方案，高質量的初始掩膜可以立即進行細化以獲得最終高質量的掩膜解決方案。

雖然光學鄰近校正有效提高了掩膜的質量，但掩膜中仍然可能存在可能導致開路或短路失敗的熱點區域，這是由於光刻變異引起的。為了進一步確保製造良率，在

物理驗證階段，準確的熱點檢測器是不可或缺的。我們提出了一個端到端的一階段熱點檢測框架。我們利用邊界點和中心點表示來提高分類和定位的準確性。此外，我們設計了一個特徵聚合模塊，通過橋接不同位置的特徵之間的關係，捕捉全局佈局信息。

我們進一步觀察到，熱點檢測和光學接近校正模型需要佈局結構信息，並且這兩個任務在芯片製造過程中密切相關。基於這種密切關係，我們提出將光學接近校正和熱點檢測整合到統一的深度學習模型中，將有助於兩個任務的性能。為了建立光學接近校正和熱點檢測之間的關聯，我們首先在掩膜建模技術上預訓練一個佈局理解模型，該模型有效地捕捉佈局的幾何信息，然後可以在有限的數據上輕鬆地對預訓練的模型進行熱點檢測和光學接近校正的微調。為了完全預訓練佈局理解模型（LUM），我們使用佈局生成技術創建了一個大型佈局數據集，解決了數據需求量大的問題。

根據實驗結果，我們證明了我們提出的深度學習方法能夠緩解物理驗證中出現的關鍵運行時間問題。隨著半導體製造技術的發展，佈局越來越複雜。我們希望本論文提出的框架能夠為先進的可製造性研究提供更强大的解決方案。

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Electronic Design Automation (EDA) encompasses a broad range of software tools and methodologies, which are used to design and verify electronic systems, particularly integrated circuits (ICs). EDA tools are essential in the semiconductor industry, enabling the design, simulation, verification, and manufacturing of complex ICs with billions of transistors. The EDA process typically involves several stages, including system-level design, logic synthesis, physical design, and verification.

Physical verification is a critical stage in the EDA workflow, ensuring that the designed circuits are manufacturable and meet all specified constraints. The physical verification involves multiple steps, including design rule checking (DRC), optical proximity correction (OPC), hotspot detection (HSD), etc. For example, DRC ensures that the layout conforms to the manufacturing rules defined by the foundry, such as minimum spacing between features and layer-specific constraints. The key to physical verification is the ability to detect and correct errors that could lead to manufacturing defects or suboptimal performance. This stage is increasingly challenging due to the miniaturization of semiconductor devices and the complexity of modern IC designs. In recent years, machine learning (ML) has emerged as

a powerful tool in addressing the challenges of EDA, particularly in the area of physical design and verification. To be specific, ML models can be applied to many predictive EDA tasks, which provide early and rapid predictions for chip optimization. By leveraging ML model predictions, designers can reduce reliance on time-consuming EDA tools and directly estimate the effects after running the EDA tools. Based on ML predictions, designers can promptly adjust design parameters and further improve the design efficiency.

1.1 Challenges

Although significant progress has been made, there are still many challenges, primarily in the following three aspects:

Firstly, as semiconductor technology advances and integrated circuits become smaller, the number of rules has significantly increased, from a few hundred in $65nm$ nodes to thousands of rules in $7nm$ nodes. Moreover, some design rules are inherently complex, often involving intricate conditions, which can easily lead to misunderstanding. Currently, the rule-making process is mainly manually conducted, which further compounds the issue. As the number of design rules continues to grow in advanced technology nodes, the manual workload involved in rule-making becomes more time-consuming. Consequently, there is an urgent need to alleviate the manual burden in the rule-making procedure.

Secondly, due to the proximity effect during the lithography process, RETs are developed to modify the photomasks in the lithographic processes, which compensates for limitations in the optical resolution of the projection systems. Mask optimization is one of the typical RETs. However, the increasing complexity of mask patterns presents challenges in terms of runtime overhead for mask optimization. Typically,

conventional mask optimization is solved by iterative methods like gradient descent, which require numerous iterations to converge. Therefore, the optimization process requires significant runtime overhead, particularly for advanced technology nodes like extreme ultraviolet (EUV) lithography. This inefficiency hinders the practical application of ILT and highly demands more efficient solutions.

Thirdly, although RETs effectively improve the mask quality, there may still exist hotspots that can potentially lead to open or short failures, which are caused by lithography variations. By performing hotspot detection before lithography simulation, potential hotspot regions can be identified early in the design flow. Such a look-ahead mechanism allows design teams to address these hotspots by applying design rule modifications or layout optimizations in advance. Early detection of hotspots enables designers to mitigate potential issues and improve the overall manufacturability of the chip. However, hotspot detectors face the challenge of relying on limited and specific hotspot pattern libraries, which hinders the design of efficient hotspot detectors. The limitations in the generality of the hotspot pattern libraries become significant factors that impede the development of efficient hotspot detection methods.

1.2 Thesis Overview

This thesis aims to contribute to the field of chip manufacturing by investigating four works that address critical aspects of design automation and system integration. These works delve into topics such as DRC, OPC, and HSD. By examining the findings and insights presented in these works, we can identify potential future directions and areas of research that hold promise for pushing technology innovation forward.

In Chapter 3, we propose the first DRC script generation framework, leveraging a deep learning-based key information extractor to automatically identify essential arguments from rules and a script translator to organize the extracted arguments into executable DRC scripts. We further enhance the performance of the extractor with three specific design rule generation techniques and a multi-task learning-based rule classification module.

In Chapter 4, we explore the challenges associated with traditional ILT algorithms and propose a new deep learning-based approach to improve the efficiency of mask optimization. Different from previous works that employ black-box models with stacked convolutional layers, our proposed L2O-ILT framework unrolls the iterative ILT optimization algorithm into a learnable neural network with high interpretability, which can generate a high-quality initial mask for fast refinement.

In Chapter 5, we address the limitations of traditional hotspot detection methods based on CNNs, which cannot capture the global layout information. However, the global layout information proves to be critical for hotspot detection. To solve this issue, we introduce a comprehensive one-stage hotspot detection framework that enhances both classification and localization accuracy through the utilization of corner and center representation. Our framework incorporates a feature aggregation module that offers a novel approach to aggregating diverse features and generating enhanced features.

In Chapter 6, inspired by large pre-trained models in deep learning, we present a customized training scheme called domain-crossing masked layout modeling. By generating a large layout dataset and incorporating lithography process awareness, this scheme effectively pre-trains the layout understanding model (LUM) on layout geometric information. The fine-tuned LUM achieves remarkable performance in both OPC and HSD tasks. This chapter also emphasizes the importance of data

generation techniques and the integration of domain knowledge in training deep learning models for layout understanding.

Chapter 7 makes a summarization and proposes a series of future study directions.

Chapter 2

Literature Review

2.1 Design Rule Checking

DRC is a fundamental step in the EDA process, ensuring that integrated circuit layouts conform to the manufacturing constraints specified by semiconductor foundries. DRC is crucial for the manufacturability, performance, and reliability of ICs, playing a pivotal role in the physical verification phase of chip design. The primary objective of DRC is to verify that a given IC layout adheres to a set of predefined rules that dictate the minimum allowable distances, widths, and other geometric and electrical requirements. These rules are necessary to prevent issues such as short circuits, open circuits, and electromigration. The typical DRC process involves the following steps: (1) Rule definition: Foundries provide detailed design rules that describe the geometric and electrical constraints for each layer of the IC. (2) Layout analysis: The IC layout is analyzed against these rules using DRC tools, which scan the layout geometries to detect violations. (3) Violation reporting: Any discrepancies between the layout and the design rules are reported as violations. (4) Correction: Designers must correct these violations so that the layout complies with all design

rules. However, with the continuous scaling-down of technology nodes and the increasing number of rules, the DRC process suffers from severe runtime overhead issues.

To improve the DRC process, some methods optimize the DRC from a parallel computing perspective. Relying on parallel computing, these methods can achieve simultaneous execution of multiple tasks or instructions across multiple processing units within a layout. Therefore, it can fully utilize the computation resources to improve the DRC efficiency. For example, George *et al.* [1] develop an algorithm where the layout design is partitioned into vertical slices, allowing individual slices to be checked separately on different processors. Different from [1], Nandy *et al.* [2] introduce a distributed approach by exploiting spatial or layer independence. This parallel algorithm partitions the layout into sub-layouts and allocates DRC tasks to idle processors in a distributed computing environment to achieve load balancing. Gregoretti *et al.* [3] consider the hierarchical information of layout and exploit cell-level parallelism computation. With the emergence of more powerful computing resources, He *et al.* [4] propose a highly efficient DRC engine that develops an adaptive row-based partitioning scheme and relies on GPU parallel processing, successfully achieving efficient DRC on hierarchical layouts.

The second category of methods considers optimizing the DRC by designing suitable data structures. Studies are conducted on the selection of appropriate data structures and algorithms to handle layout data. Bentley *et al.* [5] bring forth an algorithm for reporting intersections of rectangles, which is extracted from the DRC process. It utilizes a two-stage approach involving edge intersections and enclosing pairs of rectangles. [6] develops the concept of hinted quad-trees as a data structure for efficient neighbor searching in geometry design-rule checking, achieving high performance and reasonable storage usage. Marple *et al* [7] introduce a novel layout

system called Tailor, which employs corner stitching as a data structure for efficient access and search operations. As a result, Tailor allows for integrated tools to operate directly on the layout hierarchy, resulting in a fast and efficient design system. R-tree [8], a dynamic index structure for efficient spatial searching, can be used to represent layout based on bounding box intervals. Using R-tree allows for quick retrieval based on the spatial locations.

In recent years, approximation methods have been employed to improve the runtime efficiency of certain tasks at the expense of sacrificing result accuracy. Francisco *et al.* [9] develop a deep learning-based method using a CNN as a feature extractor. The proposed framework [9] can be used to detect and identify multiple design rule violations in layout designs, achieving faster detection and higher accuracy compared to traditional Boolean checkers. During the placement or routing stage, ML is widely used to predict DRC hotspots and estimate the number of DRVs, without the need for precise violation identification. Zeng *et al.* [10] utilize random forest as an effective and efficient model for predicting DRC hotspots in VLSI layouts, and introduce the SHAP value as a metric for explaining individual DRC hotspot predictions. Routnet [11] adopts CNN for routability prediction in mixed-size designs, accurately forecasting overall routability and predicting hotspots. R. Islam *et al.* [12] apply the ensemble random forest algorithm to predict DRC violations before global routing. A. F. Tabrizi *et al.* [13] propose to extract features from a placed netlist and feed to a neural network to detect short violations in detailed routing.

2.2 Mask Optimization

As feature sizes decrease in advanced nodes, the influence of the proximity effect becomes more noticeable, leading to a decrease in manufacturing yield. To tackle this problem, a resolution enhancement technique called OPC has been devised. OPC aims to compensate for these effects by modifying the original design layout to improve the final printed image. This technique plays a crucial role in guaranteeing pattern quality.

Typical OPC methodologies include model-based approaches [14, 15, 16] and ILT-based methods [17, 18, 19, 20, 21, 22]. For model-based OPC, the edges of polygons in the mask are first divided into segments, and these edges are moved under the guidance of the lithography simulation model [14]. Different from model-based methods, ILT-based methods represent the mask as a pixel-wise function [17, 19, 18, 20, 23] or level-set function [21, 22, 24, 25]. Then, the OPC process is modeled as an inverse problem, which can be effectively solved by optimizing the misfit between the printed image on the wafer and the target pattern. Compared with model-based methods, ILT-based methods optimize the mask within a larger solution space and thus achieve better performance. ILT algorithms usually adopt iterative methods such as gradient descent to optimize the objective function. For example, Gao *et al.* [17] simultaneously optimize the design target and minimize the process window based on the gradient descent technique. It achieves improved contour fidelity and considers manufacturing variability for better mask optimization. Ma *et al.* [23] introduce a unified framework for simultaneous layout decomposition and mask optimization, achieving faster optimization and improved design quality by employing gradient-based approaches and discrete optimization techniques. The proposed unified model also successfully addresses the inconsistency between

layout decomposition and mask optimization.

Although ILT has shown satisfactory performance on mask optimization [26, 27, 28], the shrinking size of the technology node and increasing complexity of mask patterns pose significant challenges to the runtime overhead. There have been many exciting explorations of ILT acceleration in recent years, and these works can be generally divided into two categories. (1) The first one is to design GPU-accelerated algorithms by fully utilizing the massive computing resources in GPUs. For example, Yu *et al.* [21] rely on the CUDA toolkit to implement a GPU-accelerated Fourier transform algorithm, accelerating a critical and time-consuming step in the lithography simulation model. Inspired by the multigrid method, the proposed architecture in [29] involves a lithography simulation scheme that operates at multiple resolutions. The proposed scheme simulates the layout pattern from coarse to fine resolution, effectively achieving ILT acceleration. (2) The second one is to learn the whole ILT solver using stacked convolutional layers. Related methods [19, 20, 22, 18] utilize a pre-trained CNN-based generation model such as GAN [30, 31] or U-Net [32, 33] to quickly approximate an initial mask solution of the test target and then conduct further refinements on the initial mask to improve the solution quality. We summarize these methods as "generative ILT". For example, Yang *et al.* [34] propose GAN-OPC based on a GAN model combined with lithography-guided techniques, which achieve improved mask printability and reduce computational resources. To achieve full-chip scale OPC, DAMO [35] utilizes a deep lithography simulator and a deep mask generator to optimize full-chip mask patterns, outperforming state-of-the-art solutions in both academic and industrial settings. To further improve the ILT efficiency, Jiang *et al.* [36] design an end-to-end framework called Neural-ILT, which combines mask prediction and ILT correction in a single neural network. Following [36], Chen *et al.* [22] bring forth a deep-

neural level set technique for instant mask optimization. Different from Neural-ILT [36], Develset [22] replaces pixel-based masks with implicit level set-based representations and utilizes a neural network and a CUDA-based mask optimizer for fast convergence. To further reduce the mask complexity, A2-ILT [37] develop a GPU-accelerated ILT framework with a spatial attention mechanism, which improves the ILT quality by introducing spatial attention maps and reinforcement learning, effectively reducing the manufacturing shot count.

2.3 Hotspot Detection

During chip manufacturing, the lithography process is employed to transfer layout patterns onto silicon wafers. However, the lithography process always involves many variations, and some patterns are sensitive to these variations. The layout patterns that are sensitive to process variations are defined as hotspots [38, 39]. Hotspots always cause potential open or short-circuit failures, which will undoubtedly reduce manufacturing yields.

To ensure that layout designs can be accurately printed, it is essential to have an efficient and precise hotspot detection system. The first category of the hotspot detection method is based on lithography simulation [40, 41]. These methods typically sample a series of points on the optimized masks of the layout pattern and then measure edge placement errors (EPEs). The locations with unacceptable EPEs are regarded as hotspot regions. The main drawback is that they cannot achieve end-to-end hotspot detection since they always require mask optimization results, causing extra time overhead. Moreover, the calculation of EPE is not straightforward, and due to the varying complexities of layouts, the number of sampling points differs significantly, making it challenging to perform parallel computations. Therefore, it

is not an efficient hotspot detection method to find the hotspot after the lithography simulation. Besides, since such a detection process involves sampling, the accuracy of hotspot detection is also influenced by the sampling frequency. Higher sampling frequencies result in higher detection accuracy but unsatisfactory efficiency.

To improve efficiency, various practical and efficient hotspot detection methods have been developed to overcome these limitations. The main idea is to directly detect hotspot regions from the target patterns instead of analyzing the mask patterns after the time-consuming lithography simulation process. In this way, the overall runtime can be remarkably reduced. Recent hotspot detection can be generally divided into two categories, pattern matching-based methods and machine learning-based methods. The first one relies on pattern matching techniques. These methods [42, 43, 44, 45] first take a collection of hotspot layout patterns and use them to scan over new designs to identify any matched patterns as hotspots. For example, Yao *et al.* [43] propose an efficient and scalable algorithm called range pattern matching to detect hotspots by representing them using range patterns and identifying them in a given layout. Yu *et al.* [44] develop a detection framework that extracts critical design rules and utilizes a two-stage filtering process to detect hotspots. Wen et al. [46] present a fuzzy-matching model for hotspot detection. It dynamically adjusts fuzzy regions around known hotspots and utilizes a grid reduction technique to reduce computational complexity. Although pattern matching overcomes the time-consuming issue, it fails to detect unknown hotspots since the accuracy of these methods is closely related to the completeness of the constructed hotspot datasets. When confronted with unknown hotspot patterns, it is difficult for these methods to match the unknown patterns with any hotspot patterns collected in the datasets. Therefore, pattern matching-based methods always have poor generalization ability.

In recent years, the development of machine learning has provided useful tech-

niques for hotspot detection. Different from the pattern matching-based methods, machine learning-based hotspot detection frameworks have successfully shown their ability to identify both known and unknown hotspot patterns with the help of generalized feature extractors. In general, machine learning-based methods formulate the hotspot detection task as a binary classification problem, which can be effectively tackled by neural networks. The neural network models employed have also undergone a transition from simplicity to complexity along with the development of machine learning. At the early stage, commonly adopted machine learning techniques for hotspot detection [47, 48, 49, 50, 51, 52, 53, 54] include support vector machine (SVM), Gaussian Process, Adaboost, etc. For example, Yu *et al.* [53] develop an accurate lithography hotspot detection method based on a combination of principal component analysis (PCA) and SVM classifier, utilizing hierarchical data clustering and data balancing techniques. It successfully enhances the performance and flexibility in adapting to new lithography processes and rules. Ye *et al.* [55] address the issue of uncertainty in hotspot detection and introduce a Gaussian process assurance to provide confidence in each prediction. Matsunawaa [51] *et al.* utilize a simplified layout feature and a robust classifier based on the probability distribution function of layout features.

However, there are still some issues with conventional machine learning-based methods. The main drawback is that their performances are determined by the manually crafted feature extractors. The advancement of deep neural networks (DNNs) has spawned powerful techniques to further improve hotspot detection performance [56, 57, 58, 59, 60]. Instead of relying on handcrafted features, DNNs extract hierarchical representations of the input data through the successive layers of neurons. This ability to learn features eliminates the need for manual feature engineering, which can be extremely time-consuming. For example, Yang *et al.* [56]

propose a deep learning framework for layout hotspot detection, utilizing feature tensor generation to preserve spatial information and a biased learning algorithm to improve detection accuracy. Geng *et al.* [58] design an end-to-end detection framework that combines layout feature embedding and hotspot classification using an attention-based deep convolutional neural network. To achieve faster detection speed than previous detection framework, Chen *et al.* [59] get inspiration from the object detection problem in computer vision and propose R-HSD, which can locate the regions of multiple hotspot locations on larger clips. Instead of representing the layout using an image, Sun *et al.* [61] propose a detection approach using a modified GNN and a novel graph representation scheme to transfer a layout to a graph. The feature extraction is more efficient compared with CNN-based extractor and therefore, it achieves significant speedup while maintaining accuracy.

Chapter 3

Efficient Design Rule Checking Script Generation via Key Information Extraction

3.1 Introduction

Design rule checking (DRC) is an important step in electronic design automation (EDA) flow. It checks whether a layout conforms to a set of design rules, which specify certain geometric and connectivity restrictions to ensure sufficient process window in manufacturing and guarantee the proper functionality. Figure 3.1 sketches the complete DRC flow, consisting of two phases. (1) Rule making: manufacturers first specify the essential design rules based on their manufacturing capability and then convert them into the executable DRC scripts manually, which is illustrated in the first phase of Figure 3.1. (2) Rule checking: these scripts are provided to a design rule checker, such as KLayout [62] and LayoutEditor [63], to verify the correctness of the layout design.

Modern DRC process is time-consuming and error-prone mainly in three aspects. (1) With the rapid development of semiconductor technology and the shrinking size of integrated circuits, the number of rules has grown from a few hundred in 65nm nodes to thousands of rules in 7nm nodes. (2) Different checkers require different script languages, resulting in additional efforts to re-implement and transfer scripts between checkers, as shown in Figure 3.2. (3) Some design rules can be very complicated, e.g., with complex conditions, which may easily lead to misunderstanding.

In the past few years, advanced deep learning techniques have spawned many frameworks for effectively and efficiently solving EDA problems, including physical design [64, 65, 66, 67], mask synthesis [68, 19, 69, 70], physical verification [71, 58, 72, 73], testing [74, 75, 76], etc. The literature has also explored to accelerate the rule checking phase in DRC with deep learning techniques and demonstrated promising efficiency with acceptable accuracy. For example, A. F. Tabrizi *et al.* [13] proposed to extract features from a placed netlist and feed to a neural network to detect short violations in detailed routing. R. Islam *et al.* [12] developed the ensemble random forest algorithm to predict DRC violations before global routing.

Despite the previous efforts to accelerate the rule checking phase with deep learning techniques, the rule making phase is still done manually, which requires more and more turn-around time with increasing numbers of design rules in advanced technology nodes. In preliminary work, we argue that it is of great importance to ease the manual workload in the rule making procedure. In accordance with this argument, we have proposed a DRC script generation flow (DRC-SG) in [77] where the rule making problem is formulated into a natural language processing task, which can be conducted automatically by computers. To the best of our knowledge, [77] is the first work to investigate methods for efficient DRC script generation.

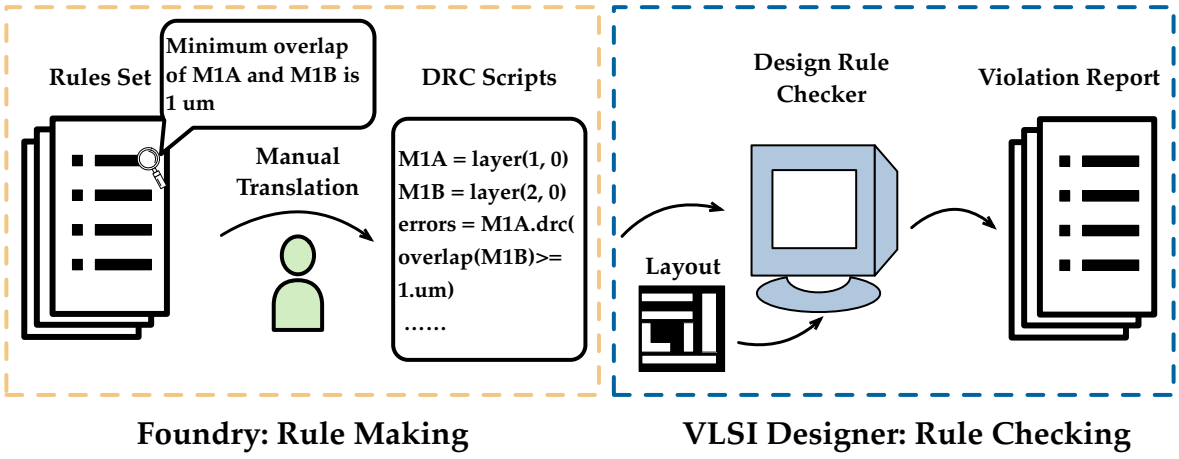


Figure 3.1: *Entire design rule checking flow.*

KLayout Script

```
M1A = layer(1, 0)
M1B = layer(2, 0)
errors = M1A.drc(
overlap(M1B)>=
1.um)
```

Guardian Script

```
Layer:
  M1A (1, 0),
  M1B (2, 0)
InDistance:
  layer1 = M1A,
  layer2 = M1B,
  limits >= 1
```

LayoutEditor Script

```
M1A = 1
M1B = 2
layout->drcTool->
minimumOverlap
(M1A, M1B, 1)
```

Figure 3.2: *Script languages vary in different checkers.*

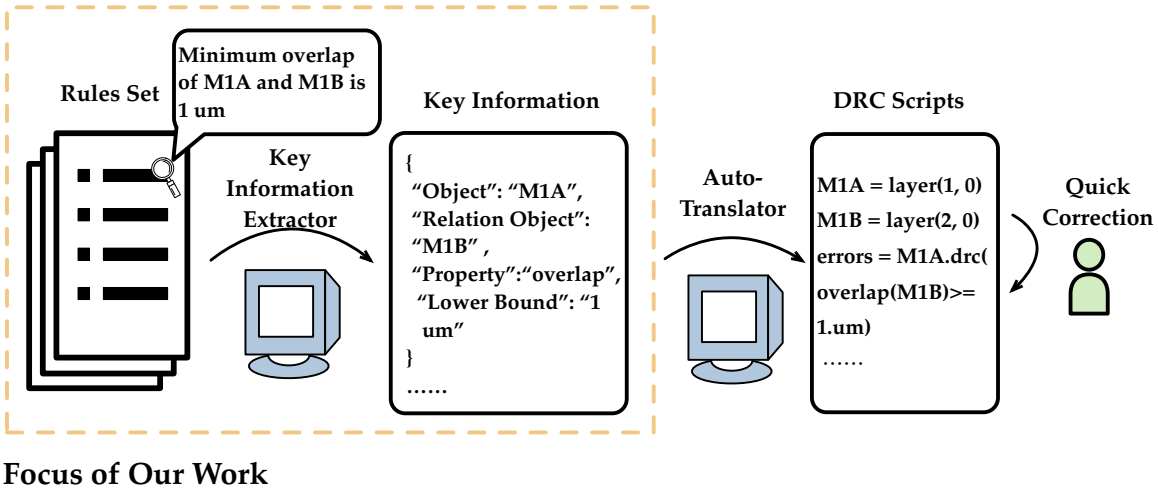


Figure 3.3: *Our proposed automatic DRC script generation (DRC-SG) flow. This is the optimization aiming at the rule making phase in Figure 3.1.*

As shown in Figure 3.3, the proposed flow in [77] relies on an automatic DRC script generation engine which consists of two stages: (1) a deep learning-based key information extractor to automatically identify the essential arguments of rules, and (2) an automatic script translator to generate scripts based on the key information extracted. The quality of the generated script is correlated to the accuracy of the key information extractor.

There are several advantages of the script generation flow in Figure 3.3. For example, with an efficient and accurate key information extractor, most generated rules are correct, so process engineers only need to do quick verification and make minor corrections on a few rules, which can significantly reduce the manual efforts and the turn-around time. In addition, our flow is highly adaptive to different design rule checkers with different script languages, as the key information extractor is a common component, and the script translator can be easily modified to accommodate new language formats. However, there still exist some defects in our preliminary design. For example, our previous flow does not fully leverage

the rule information provided by the process design kit, such as the category of each design rule. The extractor will have stronger rule understanding abilities if the model is trained to recognize the rule type. In addition, the imbalance issue of the dataset used for training our key information extractor is not considered. It is acknowledged that an imbalanced dataset will harm the performance of a deep learning-based model, which may cause the model to be biased towards the major class in the dataset. In this chapter, we solve such an issue by optimizing the loss function. The main contributions are summarized as follows:

- We propose an efficient DRC script generation flow and design dedicated deep learning techniques based on the state-of-the-art natural language processing model to accurately extract key information from design rules. Our proposed flow can be flexibly applied to different checkers.
- We develop data generation techniques based on the special language structures of design rules to expand the dataset, overcoming the dilemma of lacking design rule data for academic research.
- We build up a rule type prediction head for the extractor based on the multi-task learning paradigm to further improve the accuracy of the extracted information.
- A weighted cross-entropy loss function is customized for our key information extractor to overcome the imbalance issue from the training dataset.
- Experimental results on 7nm technology node demonstrate that our extractor achieves 91.1% precision and 91.8% recall on the key information extraction task. Besides, it only takes 5.46ms on average for our flow to generate the script of a single design rule.

The rest of this chapter is organized as follows. Section 3.2 introduces the problem formulation and terminologies related to this work. Section 3.3 illustrates the details of the preliminary work, including methods for rule data generation, and components of our proposed DRC script generation (DRC-SG) flow. Section 3.4 describes the techniques to enhance the performance of the preliminary DRC-SG flow. Section 3.5 shows the benchmarks and the experimental results, followed by the conclusion in Section 3.6.

3.2 Preliminaries

3.2.1 Design Rule Key Information Extraction

Extracting key information from natural language design rules is critical in our proposed script generation flow. It can be converted to such a problem that a specific word should be classified into a particular category, termed as a semantic role, such as the property to be checked or a specified minimum value. In this way, the problem can be considered as a word classification problem. We provide two examples as shown in Figure 3.4a and Figure 3.4b to illustrate the extraction process, where one rule is a simple rule and the other rule is relatively complex. All specified semantic roles will be further explained in Section 3.3.1. After finishing the word classification task, the categories and related words can be paired and then stored into a data structure, which is exactly the key information extracted from design rules. The following script translator simply organizes the extracted information into the final scripts; therefore, the accuracy of the generated scripts mainly depends on the extractor performance.

To quantitively evaluate the extractor performance, we adopt the widely used

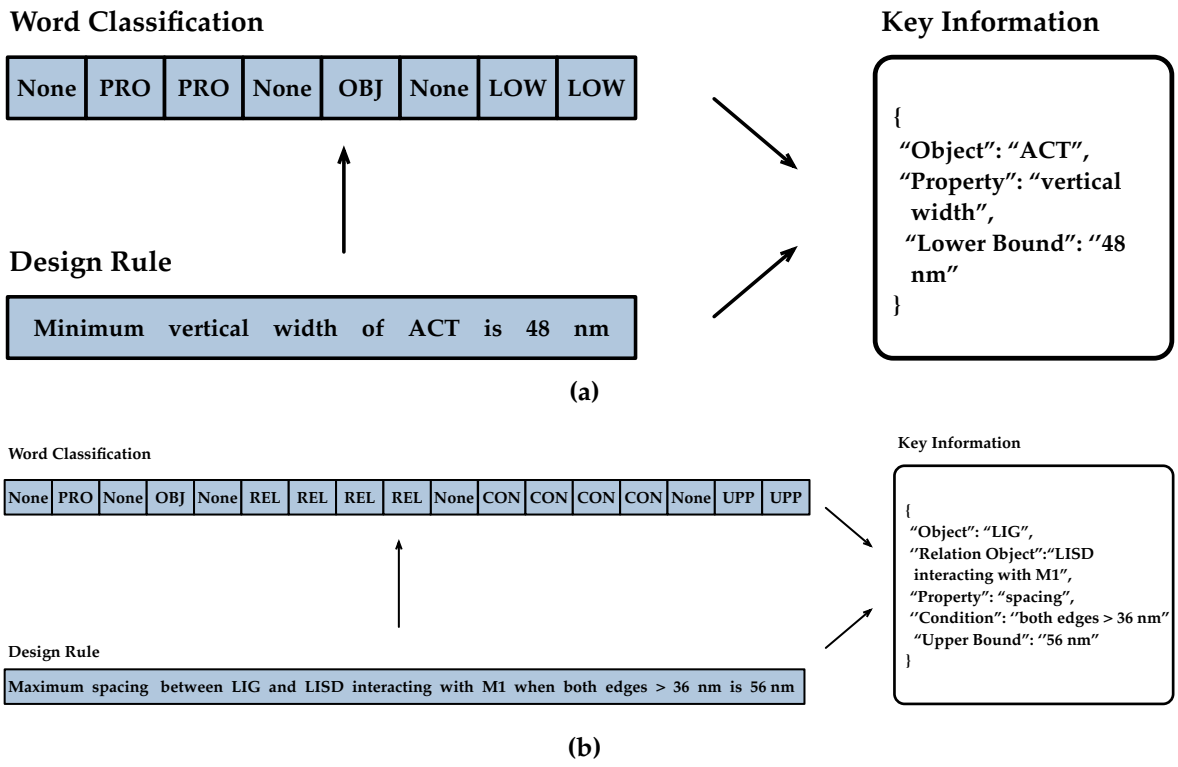


Figure 3.4: *Key information extraction process on (a) simple rule, (b) complex rule. (PRO means Property, OBJ means Object, LOW means Lower Bound and CON means Condition.)*

Table 3.1: *Confusion Matrix*

		Prediction	
		Positive	Negative
Actual	Positive	TP	FN
	Negative	FP	TN

metrics in multi-class classification problem including precision, recall and F1 score. In order to illustrate these metrics clearly, we first give the confusion matrix as shown in Table 3.1, where the rows present actual classes while columns show the prediction results. For example, given a word belonging to a specific semantic role category `S`, TP means that the category of the word is correctly predicted while FN means that the word is predicted as other categories instead of `S`. Based on the confusion matrix in Table 3.1, we give the definition of each metric as follows:

Definition 1 (Precision)**.** *Precision describes the proportion of positive predictions that is actually correct, formulated as: Precision* $= \frac{TP}{TP+FP}$.

Definition 2 (Recall)**.** *Recall describes the proportion of actual positive samples that is correctly classified, formulated as: Recall* $= \frac{TP}{TP+FN}$.

Definition 3 (F1 Score)**.** *The F1 score is the harmonic mean of the precision and recall, formulated as: F1* $= 2 * \frac{Precision*Recall}{Precision+Recall}$.

3.2.2 Transformer and BERT

Recently, Transformer [78] has made much progress in sequence-to-sequence tasks [79, 80, 81]. BERT [79] is one of the most famous models built with the Transformer encoder and has been widely used as a backbone to extract features from sentences

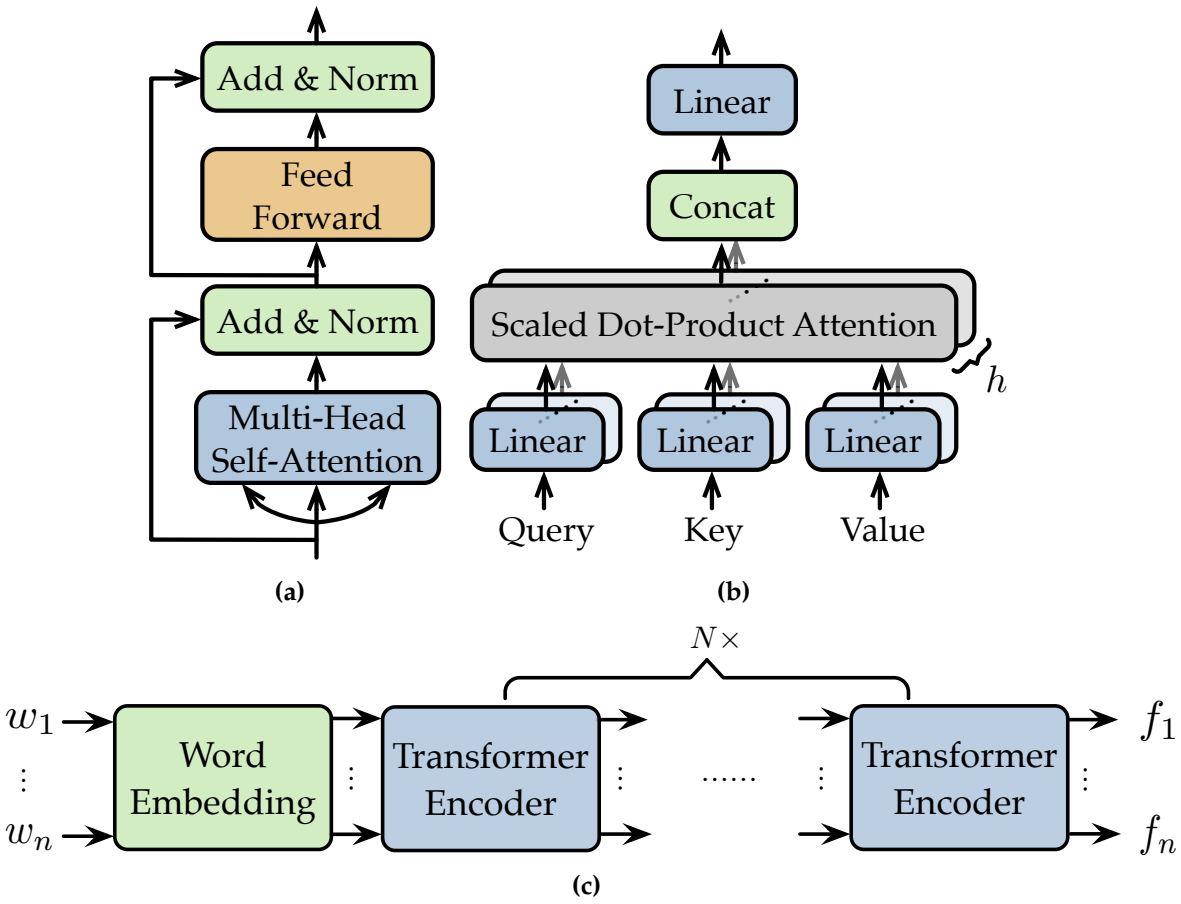


Figure 3.5: *(a) Transformer Encoder. (b) Multi-Head Self-Attention. (c) BERT.*

to solve many natural language processing problems such as Question Answering [82], Machine Translation [83], etc. To illustrate BERT [79], we first introduce the structure of Transformer encoder.

As shown in Figure 3.5a, Transformer encoder consists of multiple layers, of which the most important one is the multi-head self-attention, allowing the model to attend to information at different positions globally [78]. Given the input representation of a sequence $\{\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_n\}$, Transformer encoder first packs the sequence as a matrix, represented as $\vec{X} \in \mathbb{R}^{n \times d_m}$, where n is the length of the sequence and d_m is the dimension of each element. Then, the multi-head self-attention layer projects the input matrix $\vec{X}$ onto three different subspaces, which can be represented as:

$$\{\vec{Q}, \vec{K}, \vec{V}\} = \{\vec{X}\vec{W}^Q, \vec{X}\vec{W}^K, \vec{X}\vec{W}^V\}, \tag{3.1}$$

where $\vec{W}^Q$, $\vec{W}^K$ and $\vec{W}^V \in \mathbb{R}^{d_m \times d_m}$ are three projection matrices, which project input matrix $\vec{X}$ onto $\vec{Q}$, $\vec{K}$ and $\vec{V}$ respectively. $\vec{Q}, \vec{K}, \vec{V}$ are called *query*, *key* and *value* as named in Transformer [78]. The output of multi-head self-attention layer can be formulated as:

$$\text{MultiHead}(\vec{Q}, \vec{K}, \vec{V}) = \text{Concat}\left(\vec{H}_1, \ldots, \vec{H}_h\right)\vec{W}^O, \tag{3.2}$$

where $\vec{H}_i, i \in \{1, 2, \ldots, h\}$ is the output of a single scaled dot-product attention head as shown in Figure 3.5b and h is the number of heads. The multi-head self-attention layer concatenates all the outputs $\vec{H}_i, i \in \{1, 2, \ldots, h\}$ from different heads and then reduces the high dimension feature $\text{Concat}\left(\vec{H}_1, \ldots, \vec{H}_h\right)$ to low dimension via another projection matrix $\vec{W}^O$. To illustrate the dimension of $\vec{H}_i$ and $\vec{W}^O$, we first

give the formulation of $\vec{H}_i$ as follows:

$$\begin{aligned}\vec{H}_i &= \text{Attention}\left(\vec{Q}\vec{W}_i^Q, \vec{K}\vec{W}_i^K, \vec{V}\vec{W}_i^V\right) \\ &= \text{softmax}\left[\frac{\vec{Q}\vec{W}_i^Q\left(\vec{K}\vec{W}_i^K\right)^\top}{\sqrt{d_k}}\right]\vec{V}\vec{W}_i^V.\end{aligned} \tag{3.3}$$

For each attention head, the original input $\vec{Q}, \vec{K}, \vec{V}$ are further projected onto different subspaces via projection matrices $\vec{W}_i^Q, \vec{W}_i^K \in \mathbb{R}^{d_m \times d_k}, \vec{W}_i^V \in \mathbb{R}^{d_m \times d_v}$ so that different heads deal with different input to learn richer information [78]. The attention head then computes the similarity between projected *query* and *key* via scaled dot-product and a softmax function is applied to obtain the weights on projected *value*. As calculated in Equation (3.3), the output of each attention head $\vec{H}_i$ is a $d_v \times d_m$ matrix. The concatenated matrix Concat $\left(\vec{H}_1, \ldots, \vec{H}_h\right)$ is a $n \times hd_v$ matrix and $\vec{W}^O$ is a $hd_v \times d_m$ projection matrix.

As for BERT, the architecture is shown in Figure 3.5c, which is based on stacked Transformer encoder blocks [78] and hence incorporates the superiority of multi-head self-attention. In addition, another significant advantage of BERT [79] is that it has been fully pretrained by two complex tasks, Cloze and Next Sentence Prediction (NSP). In the Cloze task, some of the words from the input sentence are randomly masked, and the objective is to predict the masked words. As for the NSP task, it predicts whether one sentence is followed by the other sentence. Since these two tasks do not require any manual annotations, the model can be trained on two extremely huge datasets, BooksCorpus (800M words) [84] and English Wikipedia (2500M words). As a result, the pretrained BERT has been equipped with strong language representation ability and can be easily fine-tuned for other language tasks.

Table 3.2: *Explanations of all semantic roles defined in our work. The bold parts belong to the roles defined in their rows.*

Semantic Roles	Meanings	Examples
Object	Target layer of checking rules.	Minimum vertical width of **ACT** is 48 nm.
Relation Object	Additional layer that have relationships with target layer .	Minimum extension of GATEAB past **ACT** is 38 nm.
Property	Property to be checked of the target layer.	Minimum **vertical width** of ACT is 48 nm.
Condition	Logical conditions for particular layers.	GATEC shape bottom or top must be aligned if **distance is less than 192 nm**.
Restriction	Geometric restrictions that layers should follow.	GIL may **not bend**.
Lower Bound	Minimum value of the property to be checked.	Minimum vertical width of ACT is **48 nm**.
Upper Bound	Maximum value of the property to be checked.	Maximum distance of GATEAB to neighboring shape is **236 nm**.
Exact Value	Exact value of the property to be checked.	Exact horizontal spacing of ACT is **80 nm**.
None	Words do not belong to any above semantic roles	Vertical length **of** AIL1 **is** 58 nm.

3.3 DRC-SG Framework

In order to design a powerful key information extractor, both data and architecture design are important. In Section 3.3.1, we first consider how to label the design rules for effectively training the extractor. Then, in Section 3.3.2, we propose three design rule generation techniques to expand the rule dataset. The architecture of the extractor is illustrated in Section 3.3.3. In addition to the extractor, in Section 3.3.4, we explain how to design a rule-based translator to generate the scripts based on the extracted information.

3.3.1 Semantic Roles

As illustrated in Section 3.2.1, extracting key information from design rules is inherently a word classification problem. In our task, categories of words are determined based on their semantic roles in sentences. Design rule data for training our key information extractor is from an open-source design kit, FreePDK15 [85].

To clearly classify different words, we first clarify all essential semantic roles for rules in FreePDK15 [85].

Some prior works for semantic role labeling studies [86, 87] have defined roles for universal natural languages. However, semantic roles to be considered are relatively different for rule sentences in integrated circuit design. For example, numerical expressions are less frequent in these studies and usually not attributed to a separate category. In contrast, they exist in most design rules and are core components of the extracted key information. Moreover, semantic roles of numerical expressions are supposed to be further divided into three categories, i.e., "Lower Bound", "Upper Bound", and "Exact Value", which help adapt to different design rules flexibly as well as avoid confusion. We list all customized semantic roles for our task along with their meanings and examples in Table 3.2.

3.3.2 Rule Data Generation

Open-source design rules for academic research are relatively rare. Our training dataset, FreePDK15 [85], only contains around 130 rules. To help the key information extractor avoid overfitting and generalize better on those unseen rule data, we are supposed to expand the dataset before training.

Nevertheless, rule generation for our task is heavily restricted. On the one hand, since the extractor receives the design rule sentences, we are supposed to guarantee that all generated rules are both syntactically and semantically correct. On the other hand, as our task is a classification task, semantic role labels need to be assigned to each word, which is extremely expensive. Data augmentation is one kind of dataset expansion technique, referring to adding slightly modified copies of already existing data or newly created synthetic data from existing samples. There are many widely-used augmentation methods for image data, including rotation,

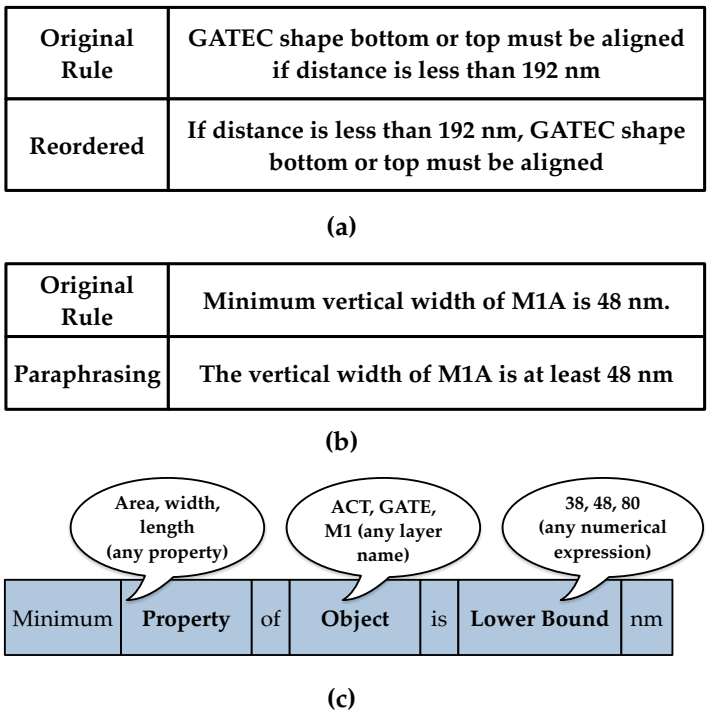


Figure 3.6: *Three rule data generation methods. (a) Word Order Adjustment; (b) Paraphrasing; (c) Template Filling.*

cropping, and noise injection, all of which are quite effective for generating new image samples. Encouraged by these methods in image tasks and considering the unique properties of our rule data, we customize three generation techniques as follows:

Word Order Adjustment. Inspired by the rotation technique for image data augmentation, we propose to adjust the word order of a rule without modifying its meaning. For example, we can settle the conditional adverbial clauses at the start or the end of the sentence, as shown in Figure 3.6a. For the human, the reordered sentence can be regarded as the same as the original one. However, from the perspective of the extractor, the input rule is a sequence $[w_1, w_2, \ldots, w_n]$ where w_i stands for a word. If the order is changed, the word of each position in the sequence will be different. Moreover, since our adjustment operation is simply changing the

position of conditional adverbial clauses and does not change any sentence content, the syntactical correctness of the generated rules can be guaranteed. In addition, adjusting the order will not affect the semantic role labels of words, and thus no extra annotations need to be done.

Paraphrasing. In contrast to word order adjustment, paraphrasing can produce a new rule that does not change the meaning but has different expressions. To be specific, we can replace some words with synonyms or change the sentence structure, e.g., from passive to active voice, and an example is given in Figure 3.6b. In order to reduce the manpower work, we first rely on a paraphrasing tool called *QuillBot* [88] and then check the correctness of the generated paraphrases by ourselves. Although paraphrasing will modify some words, which require extra annotations, there are still many words not replaced, whose semantic roles also stay unchanged. Therefore, the annotation workload can be reduced remarkably.

Template Filling. The generated design rules from the previous two methods are still confined to the meaning of the original ones, making it challenging to generate sufficient training data. To take a further step, we propose the third method, template filling. After applying the previous two generation methods, we can obtain a series of design rules with diverse sentence structures. We notice that many rules have similar structures, e.g., "Minimum width of ACT is 42 nm" and "Minimum length of GATE is 28 nm". To generate more design rule data, we first extract many templates from design rules and a representative template is "Minimum **Property** of **Object** is **Lower Bound** nm", as shown in Figure 3.6c. For example, we can fill in words related to the property, like "width", "length" and "area", in the **Property** part. And we can fill in words related to the layer name, like "ACT", "GATE" and "M1" in the **Object** part. Also, numerical value can be filled in the **Lower Bound** part. With such a template, once we associate words in

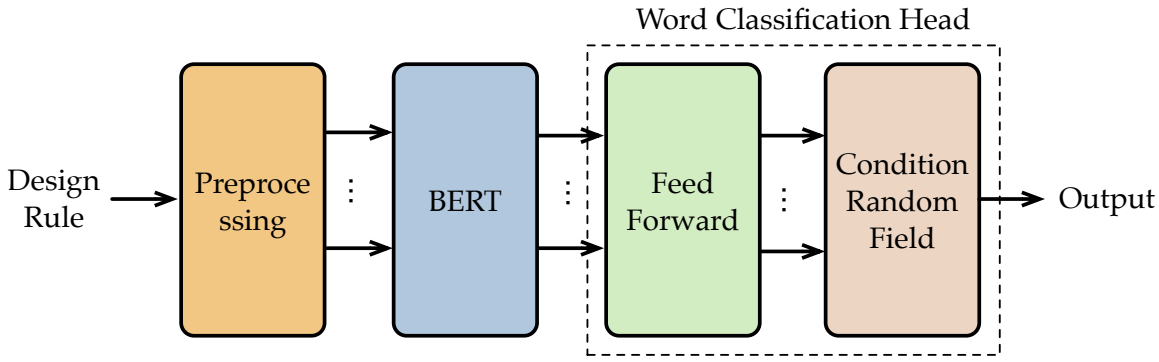


Figure 3.7: *Architecture of the key information extractor in DRC-SG.*

the bold part, the syntactical correctness of the generated rules can be guaranteed. By filling in the designed templates via different combinations, a large number of design rules can be collected for training, which benefits the generalization ability of the extractor dramatically. More importantly, we do not need to annotate the data manually since the semantic roles have been specified in advance.

After applying our customized data generation methods, we obtain 2830 new design rules, which effectively expand our rule dataset and contribute to training our key information extractor.

3.3.3 Key Information Extractor

To classify all words from design rules into their corresponding semantic roles, we build up a deep learning-based language model. The overall architecture of our framework is illustrated in Figure 3.7.

Input Preprocessing Module. Before feeding the design rules into our extractor, some preprocessing operations need to be conducted. The first one is to split the rule into a list of words for the later word classification task. Besides, since different rules vary in sentence length, we extend the word list length to a fixed value L by

padding a special word "[PAD]". The whole procedure is formulated as:

$$\text{Preprocess}(\vec{r}) = [w_1, w_2, \ldots, w_{len(\vec{r})}, \underbrace{\text{[PAD]}, \ldots, \text{[PAD]}}_{L-len(\vec{r})}], \tag{3.4}$$

where $\vec{r}$ is the input design rule. $w_i, i \in \{1, 2, \ldots, len(\vec{r})\}$ represents each word of $\vec{r}$, and $len(\vec{r})$ is its sentence length.

Backbone. Following the design paradigm of the deep learning model, we first need a backbone module to obtain a good feature representation from input rules. Determining the semantic role of each word is closely related to its sentence, and one word may have different semantic roles in different rules like "ACT" in the first and second examples in Table 3.2. Therefore, the backbone is supposed to have strong abilities to capture the context information.

Instead of designing a backbone from scratch, we adopt a powerful language model, BERT [79], as the feature extractor, which proves to have prominent feature extraction ability according to many natural language processing works like [82, 83]. As explained in Section 3.2.2, based on the self-attention mechanism, BERT is able to model interactions between any two different words in a sequence; therefore, the extracted feature of each word is closely correlated with the contexts. Besides, BERT has been fully pretrained, which means that we can fine-tune it from the pretrained parameters, notably speeding up the training procedure.

Given the word list after preprocessing, the backbone will first encode words into vectors and then feed them into stacked Transformer encoder layers. The output feature is represented as $\vec{F}^o \in \mathbb{R}^{L \times d_b}$, where d_b is the dimension of the word feature and L is the length of the input sequence.

Word Classification Head. With the purpose of classifying each word, we feed $\vec{F}^o$ into a word classifier, which is a simple feed-forward neural network composed of two fully connected layers. The output is represented as $\vec{P}^{wc} \in \mathbb{R}^{L \times N_{wc}}$, where

N_{wc} is the number of categories, and the element $\vec{P}^{wc}_{i,j}$ stands for the score of the word w_i belonging to label j.

However, such a prediction head does not take the relationships between different labels into consideration. We can force the word classification head to effectively avoid those impossible prediction sequences. For example, according to the common natural language expression habits, "Relation Object" is impossible to directly follow "Object" since there must exist some conjunctions between them. As a result, the extractor performance can be further improved by evaluating the rationality of the entire prediction sequence. To achieve this, we build a probability model, condition random field (CRF) [89], on top of the word classifier, whose parameters are a label transition matrix, represented as $\vec{K} \in \mathbb{R}^{(N_{wc}+2)\times(N_{wc}+2)}$. The element of the label transition matrix $\vec{K}_{i,j}$ describes the probability of transitioning from label i to j. Two additional states included in $\vec{K}$ stand for the "start" and "end" of the sequence. In such a case, given a design rule $\vec{r}$, the probability of a prediction sequence $\vec{y}$ is calculated from softmax function as:

$$p(\vec{y}|\vec{r}) = \frac{\exp S(\vec{r},\vec{y})}{\sum_{\vec{\hat{y}}\in\vec{Y}_{\vec{r}}} \exp S(\vec{r},\vec{\hat{y}})}, \tag{3.5}$$

where $\vec{Y}_{\vec{r}}$ represents all possible prediction sequence results given the rule $\vec{r}$. $S(\vec{r},\vec{y})$ is used to measure the score of prediction $\vec{y}$, which can be formulated as:

$$S(\vec{r},\vec{y}) = (\vec{K}_{start,\vec{y}_1} + \sum_{i=1}^{L-1} \vec{K}_{\vec{y}_i,\vec{y}_{i+1}} + \vec{K}_{\vec{y}_L,end}) + \sum_{i=1}^{L} \vec{P}^{wc}_{i,\vec{y}_i}. \tag{3.6}$$

In this way, $S(\vec{r},\vec{y})$ is able to measure the reasonableness of the label sequence itself.

Loss Function. After constructing the whole architecture, we need to specify the loss function to train our key information extractor. In the CRF module, we should maximize the groundtruth probability $p(\vec{g}|\vec{r})$, where $\vec{g}$ is the actual label sequence for the rule $\vec{r}$. Based on this maximization objective, the loss function $\mathcal{L}_{wc}$ of the

word classification head can be formulated in the negative log-likelihood format as follows:

$$\begin{aligned} \mathcal{L}_{wc} = \mathcal{L}_{crf} &= -\log p(\vec{g}|\vec{r}) \\ &= -S(\vec{r},\vec{g}) + \log(\sum_{\vec{y}\in\vec{Y}_{\vec{r}}} \exp S(\vec{r},\vec{y})). \end{aligned} \tag{3.7}$$

3.3.4 Script Translator

The script translator in the second stage of our proposed flow is used to translate the extracted key information into the DRC scripts. When transferring to other checkers, we can preserve the extractor and simply replace the translator. The translator design is similar and simple for different checkers, and here we take the Guardian checker [90] as an example.

DRC script is composed of function calling statements. When given the key information, the functions to be called mainly depend on the checking properties. To conveniently search the required function, we can pair the properties and functions together, as shown in Figure 3.8. In addition, to automatically pass the key information to the function, we also need to connect the parameters of different functions with our semantic roles. As we clearly define the fine-grained semantic roles in Table 3.2, the relationships can be easily established. For example, parameters that receive the layer name correspond to "Object" or "Relation Object", and parameters that receive the checking value correspond to "Lower Bound" or "Upper Bound" or "Exact Value".

To better illustrate the entire script translation process, we take the translation process of an overlap rule as an example, which is shown in Figure 3.8. `Layer` is a regular function for each Guardian script and receives the layer names along with their identifiers, which depend on the specific layout design. `InDistance`

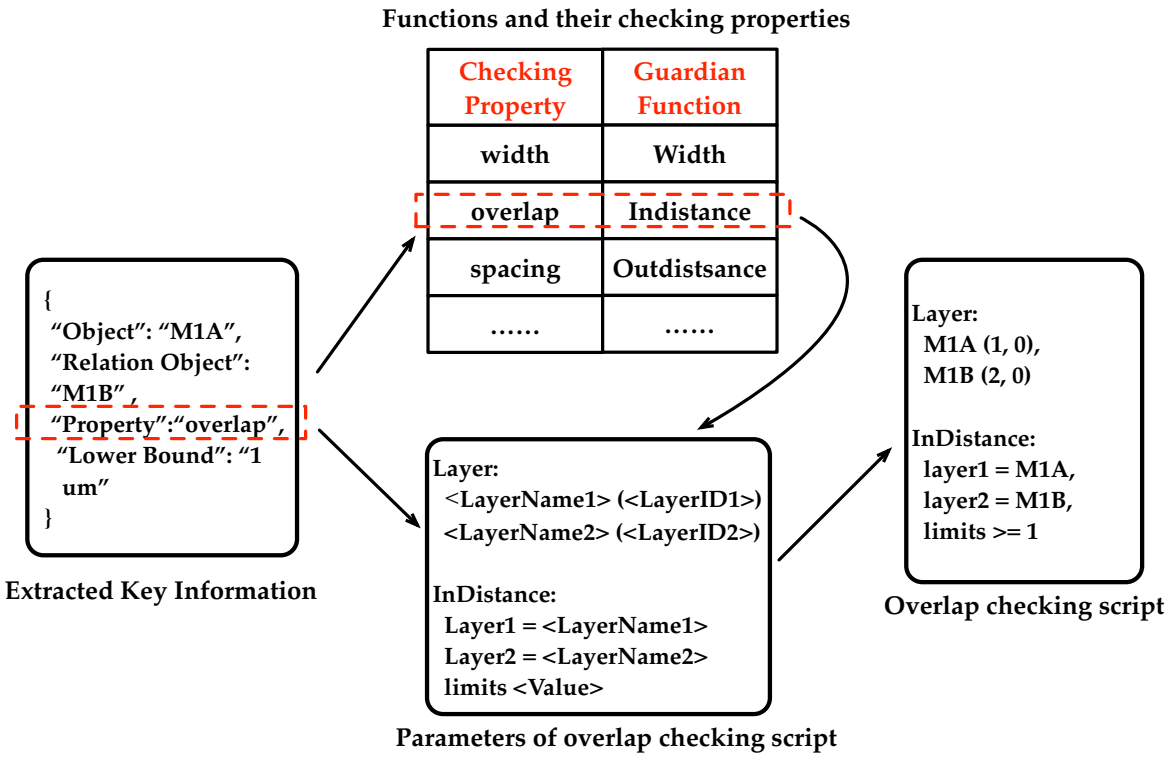


Figure 3.8: *Script translator.*

function receives two layer names and the value to check the overlap. It can be observed that all the required arguments passed to these two functions can be easily obtained from the extracted information. By automatically filling them into the corresponding placeholders, we obtain the final script for overlap checking. It can be seen that the entire translation process is very efficient.

3.4 DRC-SG 2.0

In the previous section, the preliminary DRC-SG flow has been proposed. However, there still exists some room to improve the performance of the script generation flow. For instance, the training dataset has an imbalance issue, which has a negative

impact on the performance of the key information extractor in DRC-SG. Therefore, based on the preliminary design for DRC-SG, we propose **DRC-SG 2.0** where two new techniques are introduced for further enhancement.

3.4.1 Rule Classification Head for Key Information Extractor

In addition to the concrete rule descriptions as shown in the "Examples" column in Table 3.2, the process design kit also provides another important information, rule category, which is not effectively utilized in the DRC-SG framework. Rule category is always determined by the entire rule content. Therefore, the global comprehension ability of the extractor on the design rule can be further improved if the extractor can learn to predict the rule type, which also contributes to the word classification performance.

Based on such a consideration, a new branch, rule type prediction head, is incorporated into the original key information extractor as shown in Figure 3.9. It can be seen that the rule classification head shares the same backbone with the word classification head. In order to predict the rule type, we are supposed to learn the rule feature $\vec{f}^{rc}$ by combining word features $\vec{f}_i^o, \forall i \in \{1, 2, \ldots, L\}$ output by the backbone. To achieve this, we rely on the self-attention mechanism illustrated in Section 3.2.2 and make some customizations.

Firstly, instead of utilizing the entire word features $\vec{f}_i^o, \forall i \in \{1, 2, \ldots, L\}$, a well-designed algorithm is proposed to select part of them, which is illustrated in Algorithm 1. The main idea of our selection algorithm is to leverage the probability matrix $\vec{P}^{wc}$ output by the word classification head to guide the word feature selection. $\vec{P}^{wc}$ describes the probability of each word belonging to each semantic role. We propose that words with lower probabilities belonging to "None" category are more informative, and based on this metric, we choose features of the most

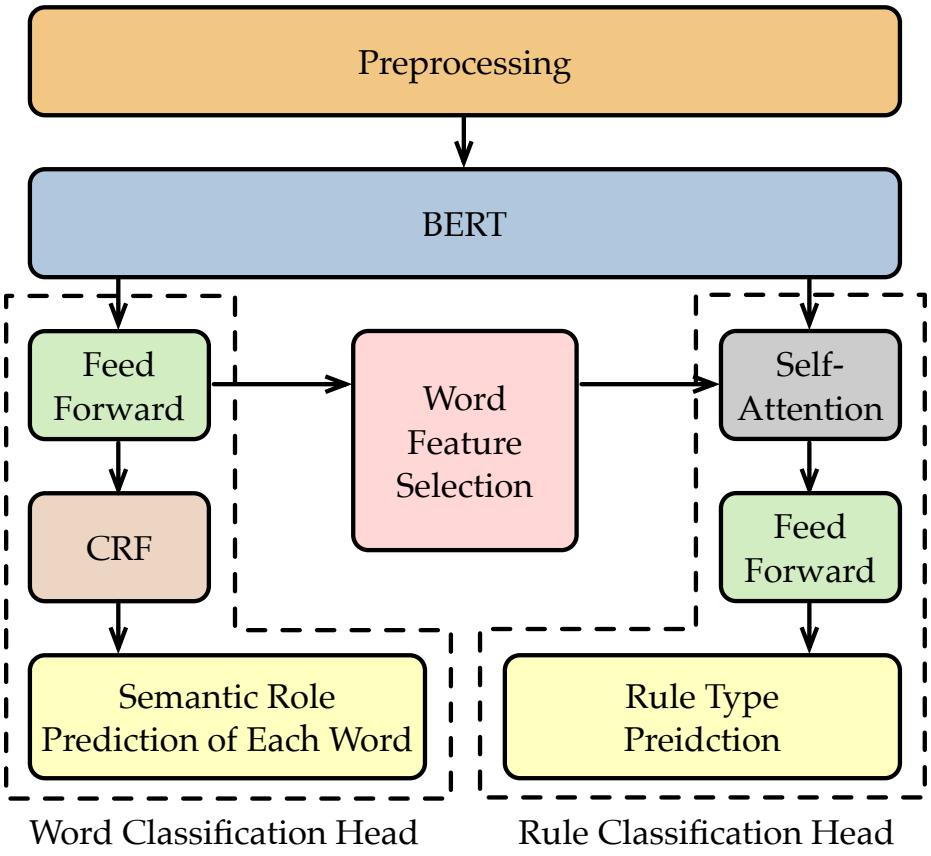


Figure 3.9: *Architecture of the key information extractor with rule prediction head in DRC-SG 2.0.*

informative k words for rule type prediction.

Algorithm 1 Word Feature Selection Algorithm

Input: Word Classification Output $\vec{P}^{wc} \in \mathbb{R}^{L \times N_{wc}}$, Feature Map $\vec{F}^{o} \in \mathbb{R}^{L \times d_b}$, Selection Number k;
Output: Feature set $\vec{F}^{k}$;

```
F^k ← Initialized to empty set;
p^None ← probabilities of all words belonging to "None" class.
SelectIdx ← indices of the minimum k probabilities in p^None;
for i ← 1, 2, ...k do
    idx ← SelectIdx[i];
    f_i^o ← the feature in the position idx of F^o;
    append feature f_i^o to F^k;
return feature set F^k with k word features.
```

Then, we introduce an extra variable $\vec{q}^{rc}$ as the *query*, and features $\vec{f}_i^k, i \in \{1, 2, \ldots, k\}$ from $\vec{F}^k$ are regarded as the *key* and *value*. Different from the original multi-head self-attention computation paradigm as formulated in Equation (3.2), we only adopt a single self-attention head, which is capable of this sub-task and also saves computation resources. In this way, the rule feature can be represented as follows:

$$\vec{f}^{rc} = \sum_{i=1}^{k} \frac{\exp(\vec{q}^{rc}(\vec{f}_i^k \vec{W}^K)^\top / \sqrt{d_b})}{\sum_{j=1}^{k} \exp(\vec{q}^{rc}(\vec{f}_j^k \vec{W}^K)^\top / \sqrt{d_b})} \vec{f}_i^k \vec{W}^V, \tag{3.8}$$

where $\vec{f}^{rc}, \vec{q}^{rc} \in \mathbb{R}^{d_b}$ have the same dimension as $\vec{f}_i^k$. $\vec{W}^K, \vec{W}^V \in \mathbb{R}^{d_b \times d_b}$ are projection matrices as explained in Section 3.2.2. Noted that $\vec{q}^{rc}$ is a learnable variable whose value is determined after extractor training.

Once the representation $\vec{f}^{rc}$ of the whole design rule is obtained, we feed it into a rule classifier, which is a neural network with three fully-connected layers. The final output is represented as $\vec{p}^{rc} \in \mathbb{R}^{N_{rc}}$, elements of which are prediction scores of all rule types, and N_{rc} represents the number of rule categories.

After introducing the rule classification head, the loss function for training our

extractor is supposed to be redesigned as follows:

$$\mathcal{L} = \mathcal{L}_{wc} + \lambda \mathcal{L}_{rc}, \tag{3.9}$$

where $\mathcal{L}_{wc}$ is the loss of the word classification head and $\mathcal{L}_{rc}$ is the loss of the rule classification head. λ is a hyperparameter to balance $\mathcal{L}_{wc}$ and $\mathcal{L}_{rc}$. Since the rule classification head conducts a multi-classification task, we utilize the conventional cross entropy loss for $\mathcal{L}_{rc}$, computed as:

$$\mathcal{L}_{rc} = -\log \frac{\exp\left(\vec{p}_t^{rc}\right)}{\sum_{i=1}^{N_{rc}} \exp\left(\vec{p}_i^{rc}\right)}, \tag{3.10}$$

where t stands for the groundtruth rule type of the input and $\vec{p}^{rc}$ is the prediction result of the rule classification head.

3.4.2 Weighted Cross-Entropy Loss for Word Classification Head

In our previous loss function design for the word classification head, we have avoided unreasonable label sequences by utilizing the CRF module, which effectively improves the word classification performance. However, another issue is the imbalance problem of the dataset, which can not be solved by CRF. To be specific, almost every design rule has words belonging to the semantic role "Object", but may not have "Relation Object" words. The concrete proportion distribution of different semantic roles in our dataset is shown in Table 3.3, also proving the imbalance issue.

We realize that such an issue is harmful to the performance of our extractor. Specifically, the extractor will be biased towards the major class in the dataset once it is trained by an imbalanced dataset. To solve this issue, we consider introducing a weighted cross-entropy loss term $\mathcal{L}_{imb}$ into the original word classification loss, formulated as follows:

$$\mathcal{L}_{wc} = \mathcal{L}_{crf} + \eta \mathcal{L}_{imb}, \tag{3.11}$$

where $\mathcal{L}_{crf}$ is still from Equation (3.7) and η is another hyperparameter to balance $\mathcal{L}_{crf}$ and $\mathcal{L}_{imb}$. The $\mathcal{L}_{imb}$ is computed via a weighted cross-entropy loss as formulated in Equation (3.12) and $\vec{w}_{\vec{g}_i}$ is the corresponding weight of each term.

$$\mathcal{L}_{imb} = -\sum_{i=1}^{L} \vec{w}_{\vec{g}_i} \log \frac{\exp\left(\vec{P}^{wc}_{i,\vec{g}_i}\right)}{\sum_{j=1}^{N_{wc}} \exp\left(\vec{P}^{wc}_{i,j}\right)}, \tag{3.12}$$

$$\vec{w}_{\vec{g}_i} = \exp\left(\frac{1 - \vec{u}_{\vec{g}_i} / \max(\vec{u})}{\alpha}\right). \tag{3.13}$$

We utilize the groundtruth label of each word to supervise the input of the CRF module, $\vec{P}^{wc} \in \mathbb{R}^{L \times N_{wc}}$, which includes scores of words belonging to each category. $\vec{g}_i$ is the groundtruth label of the i-th word and $\vec{u} \in \mathbb{R}^{N_{wc}}$ is a vector containing proportions of different categories. $\alpha \in \mathbb{R}^{+}$ is a hyperparameter to control the weight range. It can be observed that we assign larger weight $\vec{w}_{\vec{g}_i}$ to the smaller proportion category, by which we are able to balance the extractor performance on each word category. After introducing the rule classification loss $\mathcal{L}_{rc}$ and weighted cross-entropy loss $\mathcal{L}_{imb}$, the total loss of the enhanced extractor is finally represented as follows:

$$\mathcal{L} = \mathcal{L}_{wc} + \lambda \mathcal{L}_{rc} = \mathcal{L}_{crf} + \eta \mathcal{L}_{imb} + \lambda \mathcal{L}_{rc}. \tag{3.14}$$

The objective of training is to minimize the loss calculated in Equation (3.14), which can be successfully solved by Adam [91] optimizer, a widely-used gradient descent optimization algorithm.

Table 3.3: *Semantic roles distribution of dataset*

Semantic Roles	Training Set		Test Set	
	#	Percent (%)	#	Percent (%)
Object	6054	15.68	491	14.74
Relation Object	2561	6.63	181	5.43
Property	4705	12.18	300	9.01
Condition	5061	13.10	596	17.89
Restriction	1404	3.64	159	4.77
Lower Bound	2482	6.43	326	9.79
Upper Bound	1144	2.96	8	0.24
Exact Value	1430	3.70	40	1.20
None	13778	35.68	1230	36.93
Total	38619	100	3331	100

3.5 Experiment Results

3.5.1 Experimental Settings and Benchmark

We implement our entire framework with the Pytorch library[92] in Python, and test it on a platform with the Xeon Silver 4114 CPU processor and NVIDIA TITAN Xp Graphic card. The dataset used for training our key information extractor contains 2970 design rules, 2840 of which are obtained via our proposed rule generation methods and the rest are the original data from PDK15 [85]. To evaluate the performance, another design kit, ASAP7 [93], acts as the test set, which includes 200 design rules on the $7nm$ node. Due to the advanced technology node, rules in ASAP7 are more complex compared with our rules on the $15nm$ node for training, and therefore, the evaluation performance on ASAP7 will convincingly reflect the generalization ability of our framework. We summarize the statistics of the datasets in Table 3.3. It can also be observed from Table 3.3 that the "None" category words accounts for nearly 40 percent, which further proves that a key information extractor

can filter a lot of unnecessary information and contribute to the design rules scripts generation.

3.5.2 Experimental Results and Analysis

Due to the various structure of scripts for different rules, it is not convenient to directly measure the accuracy of the generated scripts. In Section 3.2.1, we discuss that the script accuracy can be reflected by the extractor performance, and we also explain that the key information extraction task is essentially a word classification task. To evaluate the comprehensive performance, we test the word classification accuracy, inference time of the whole generation process and robustness ability of the extractor.

Table 3.4: *Word classification comparison results with two other baselines*

Semantic Roles	Bi-RNN [94]			Bi-LSTM [95]			DRC-SG [77]			DRC-SG 2.0		
	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
Object	0.609	0.666	0.636	0.772	0.662	0.713	0.853	0.804	0.828	0.916	0.872	0.894
Relation Object	0.436	0.674	0.529	0.422	0.674	0.519	0.849	0.896	0.872	0.904	0.934	0.918
Property	0.879	0.970	0.922	0.899	0.953	0.926	0.892	0.900	0.896	0.955	0.997	0.976
Condition	0.786	0.768	0.777	0.670	0.757	0.711	0.818	0.838	0.828	0.900	0.938	0.919
Restriction	0.389	0.453	0.419	0.538	0.399	0.458	0.789	0.704	0.744	0.800	0.704	0.749
Lower Bound	0.947	0.871	0.907	0.960	0.883	0.920	0.967	0.907	0.936	0.987	0.908	0.946
Upper Bound	0.500	1.000	0.667	0.429	0.750	0.545	0.889	1.000	0.941	1.000	1.000	1.000
Exact Value	0.371	0.650	0.473	0.750	0.900	0.818	0.741	1.000	0.851	0.833	1.000	0.909
None	0.853	0.714	0.777	0.883	0.825	0.853	0.892	0.894	0.893	0.900	0.912	0.906
Average	0.641	0.752	0.679	0.703	0.756	0.718	0.853	0.880	0.863	**0.911**	**0.918**	**0.913**
Ratio	0.703	0.819	0.744	0.772	0.824	0.786	0.936	0.959	0.945	**1.000**	**1.000**	**1.000**

Word Classification Results. Table 3.4 summarizes the detailed comparing results in the test set. Since our preliminary work DRC-SG [77] is the first one to investigate key information extraction methods for design rules, no other state-of-the-art work in DRC area can be referred for comparison. Therefore, we further implement two baseline models, bidirectional RNN (Bi-RNN) and bidirectional

Table 3.5: *Average word classification results on different ablation settings.*

Ablation Settings				Precision	Recall	F1
Rule Classification Head	Weighted Cross-Entropy	Condition Random Field	Rule Generation			
	✓	✓	✓	0.875	0.903	0.889
✓		✓	✓	0.895	0.913	0.904
✓	✓		✓	0.877	0.905	0.891
✓	✓	✓		0.662	0.634	0.637
✓	✓	✓	✓	**0.911**	**0.918**	**0.913**

LSTM (Bi-LSTM). Similar to BERT, Bi-RNN [94] and Bi-LSTM [95] are commonly used for learning words features combined with context information and output the category prediction results, which match the objective of our extractor. The corresponding results are listed in columns "Bi-RNN" and "Bi-LSTM" The rest two columns, "DRC-SG" and column "DRC-SG 2.0" denote the methods in [77] and the framework presented in this work.

It can be seen that the result in DRC-SG averagely outperforms Bi-RNN with 21.2% and 12.8% improvement on precision and recall and 18.4% rise on F1 score. Besides, DRC-SG surpasses Bi-LSTM with an average precision, recall and F1 score of 15.0%, 12.4% and 14.5%. Compared with our preliminary work DRC-SG [77], the word classification performance is further promoted in this work. Specifically, our new proposed DRC-SG 2.0 model achieves better results with 91.1%, 91.8% and 91.3% on precision, recall and F1 score respectively.

Noted that we do not show the rule type prediction results since the rule classification head is removed during the test stage to save the computation costs. It is designed to help promote the training performance of the word classification head, and we will show its functionality in the following ablation study part.

Inference Time. In addition to the satisfactory accuracy, our flow also shows superior efficiency. We first test the inference time of the extractor on ASAP7 dataset

which contains 200 design rules. The total runtime result is 1.0s, from which we can calculate that our model averagely takes only 5.0ms to process a single rule. As for the translator, since it is simply responsible for deciding which function to call and passing the extracted key information to the function, the translation process is also high-efficient. According to our measurement, the script translator spends around 0.46ms processing one item of key information. In conclusion, by combining the extractor and translator, our framework can generate a single script in 5.46ms on average, indicating that our proposed DRC script generation flow is extremely efficient.

Compared with Manual Script Generation. We also compare the runtime of our proposed DRC-SG 2.0 framework with manual script generation. Although the key information extractor in DRC-SG 2.0 can not guarantee absolutely accurate results and requires post-correction, however, because of the high accuracy as listed in Table 3.3, most scripts will be correct and only few errors need to be rectified in real scenarios, and thus the manual workload of scripts generation is notably reduced. We test that it takes around 30 minutes to correct all the scripts generated by our DRC-SG 2.0 framework. As for manual script generation, we calculate that a single rule averagely takes 3 minutes to write its corresponding script. Therefore, given all the 200 design rules from ASAP7 dataset, people should spend around 10 hours finishing all scripts writing, which is very time-consuming. With the help of our proposed DRC-SG 2.0, the script generation tasks achieve nearly 20 times faster than manual script generation.

Robustness Analysis. Sometimes there may exist typos in natural language design rules, which may affect the accuracy of a language model. To further evaluate the performance of the extractor, we conduct the following experiments to test the robustness ability of our extractor when encountering typos. We randomly

choose 5% words from the test datasets and then replace one character of each word to simulate typos. Without retraining our extractor, we directly evaluate the performance on the new test datasets. We repeat the above procedure 10 times and the average F1 score is 88.35%, which is close to the original F1 score (91.30%). This indicates that our extractor has powerful fault-tolerant mechanism, which is also beneficial for the stability of the whole script generation flow.

3.5.3 Ablation Studies

The major work of this chapter is to design a high-performance key information extractor. To verify the benefit of our customized components in the extractor, including rule classification head, weighted cross-entropy, CRF and data generation methods, we conduct extra ablation studies. The average word classification results of different configurations are shown in Table 3.5. In the following analysis, we mainly adopt F1 score as the metric, since it is calculated from both the precision and recall and is capable of reflecting the positive effect of each component.

Results in Table 3.5 show that with the rule generation techniques, F1 score notably improves. It is because that abundant data increase the diversity, which helps prevent the extractor from overfitting and improve the generalization ability. Besides, with CRF, we further achieve nearly 2.2% improvement on F1 score, demonstrating that CRF effectively avoids unreasonable label sequence and achieves better word classification performance. With the weighted cross entropy loss, the F1 score increases 0.9% and we also notice that the positive gains are mainly from small proportion categories such as "Upper Bound" and "Exact Value", which conforms to our design motivation. In addition, with the rule classification head, F1 score improves around 2.4%, indicating that the knowledge learned in this head can be positively leveraged for the word classification task.

3.6 Summary

In this chapter, we propose an automatic DRC script generation flow. We first build up a deep learning-based extractor that efficiently recognizes essential arguments from design rules and then leverage a script translator to organize the extracted arguments into the scripts. Experimental results on 7nm technology node have confirmed the excellent performance of our framework: it only takes on average 5.46ms to generate a single rule script, which is much faster than manual generation, and our powerful extractor achieves 91.1% precision and 91.8% recall on the key information extraction task. The number of design rules keeps increasing with the development of semiconductor technology, and generating DRC scripts manually will become more and more time-consuming. We hope the framework proposed in this work can provide a preliminary solution to automate the generation of DRC scripts and help reduce the manual workload. There still exist limitations for our model to generate design rule scripts for those extremely complex rules and we will continue to find the solution in the future.

Chapter 4

L2O-ILT: Learning to Optimize Inverse Lithography Techniques

4.1 Introduction

With the continuous scaling-down of technology nodes, the proximity effect and optical diffraction are becoming non-neglectable, which seriously affects the yield of integrated circuits. Resolution enhancement techniques (RETs) are developed to reduce printing errors during the lithography process. ILT is one of the widely used RETs to compensate for lithography proximity effects by correcting mask pattern shapes and inserting assist features. Although ILT has shown satisfactory performance on mask optimization [26, 27, 28], the continuous scaling-down of the technology node and increasing complexity of mask patterns pose significant challenges to the runtime overhead.

To further accelerate the process of ILT and keep the accuracy of mask printability, we propose L2O-ILT as illustrated in Figure 4.1 in this work. The novel ILT method is inspired by the learning-to-optimize (L2O) scheme [96, 97, 98, 99, 100] in

machine learning, which aims to incorporate prior knowledge of the optimization algorithm into a learning model. Specifically, we build up an ILT-inspired learning model by unrolling the entire algorithm. The structure of our model is no longer stacking convolutional layers. Instead, each layer is customized to represent each iteration of the ILT algorithm. This representation projects the ILT problem into a hyperspace that can be more efficiently solved by deep learning algorithms. And the model training can be regarded as automatically tuning the algorithm parameters, which are hand-crafted in the conventional ILT algorithm. In addition, such a

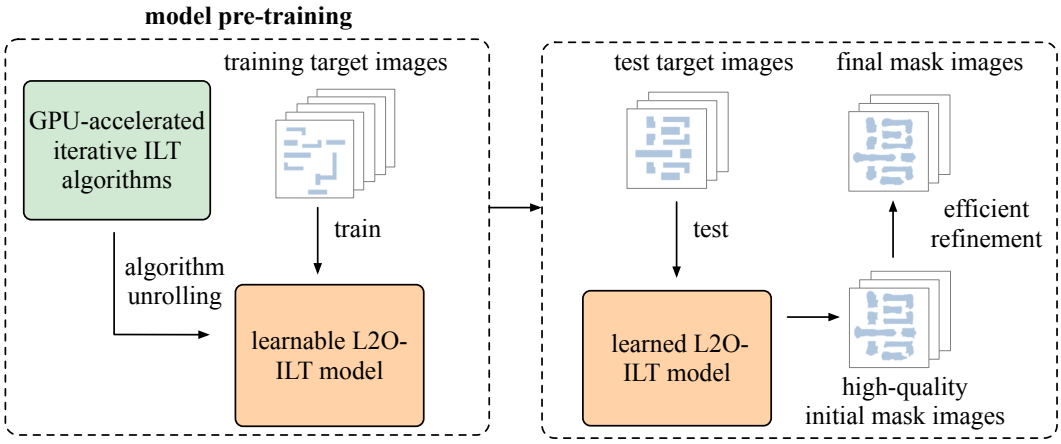


Figure 4.1: *Learning to Optimize ILT.*

model is inherently equipped with interpretability and prior knowledge of mask optimization, which will contribute to robustness and ensure a high-quality initial mask for efficient refinement. Besides, a specialized optimization mechanism called alternating optimization is designed for our model to jointly optimize the printed image under different process conditions. An adaptive solution space is developed to accelerate the convergence rate of our algorithm while saving computation resources. We summarize the contributions of this chapter as follows:

- A deep learning-based and ILT-inspired neural network called L2O-ILT is

developed, which incorporates domain-specific prior knowledge of mask optimization.

- The network architecture is designed by unrolling the ILT algorithm and modeling each iteration as a neural network layer.
- We develop an alternating optimization mechanism and an adaptive solution space method to improve the conventional ILT algorithm and further the performance.
- L2O-ILT is able to generate high-quality initial mask solutions, which can be efficiently refined. Experimental results show that our model achieves better performance on both mask printability and runtime than previous methods.

The rest of this chapter is organized as follows. Section 4.2 gives an introduction preliminaries about lithography model and inverse lithography technologies. Section 4.3 gives the detailed elaboration of the L2O-ILT model with an alternating optimization strategy and an adaptive solution space mechanism. Section 4.4 details experimental results and comparisons, followed by conclusion in Section 4.5.

4.2 Preliminaries

In this section, we will introduce the problem formulation and some preliminary knowledge related to this work.

4.2.1 Lithography Simulation Model

During the lithography process, an input mask $\vec{M}$ is first transformed through an optical projection system into the aerial image $\vec{I}$. The distribution of light intensity

at the wafer plane then undergoes development and etching processes to form the printed image $\vec{Z}$.

To simulate the lithography process, a mathematical model is proposed in [101], which is composed of two components, optical projection model and photoresist model. For the optical projection process, the Hopkins diffraction model of the partially coherence imaging system is used to approximate the projection behavior. In mathematics, the aerial image $\vec{I}$ can be obtained by convolving the mask $\vec{M}$ with a set of optical kernels $\vec{H}$, formulated as:

$$\vec{I}(x,y) = \sum_{k=1}^{N^2} w_k \left| \vec{M}(x,y) \otimes \vec{h}_k(x,y) \right|^2, \tag{4.1}$$

where "$\otimes$" represents the convolution operation, h_k is the k-th optical kernel of the optical kernel set $\vec{H}$ and w_k is the corresponding weight of the coherent system. To save the computation resources, an N_h-th order approximation to the partially coherent system is proposed in [17], represented as:

$$\vec{I}(x,y) = \sum_{k=1}^{N_h} w_k \left| \vec{M}(x,y) \otimes \vec{h}_k(x,y) \right|^2, \tag{4.2}$$

where the kernel number N_h is 24 in our work. After optical simulation, the aerial image $\vec{I}$ is input into the photoresist model with an intensity threshold I_{th}, which indicates the exposure level. And the final binary printed image $\vec{Z}$ is calculated by the following step function:

$$\vec{Z}(x,y) = \begin{cases} 1 & \vec{I}(x,y) \geq I_{th} \\ 0 & \vec{I}(x,y) < I_{th} \end{cases} \tag{4.3}$$

Following the ICCAD 2013 contest settings [102], I_{th} is set as 0.225 in our implementation.

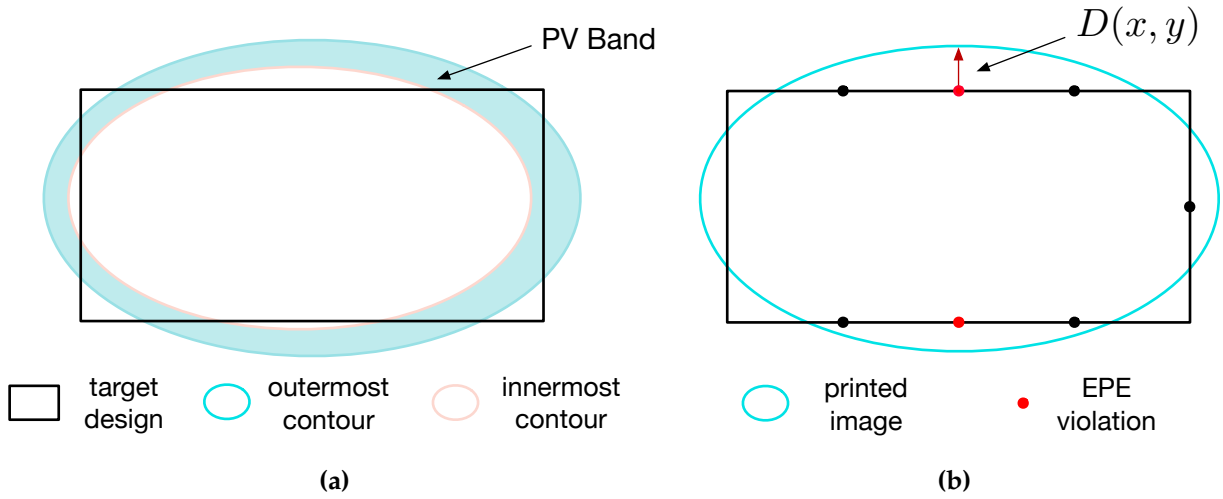


Figure 4.2: *(a) Visualization of PV Band measurement. (b) Visualization of EPE measurement.*

4.2.2 OPC Evaluation Metrics

Process Variation Band (PVB): In the real-world lithography system, process variations will cause deviations in the final printed image, leading to printing failure. Under different lithography conditions, such as focus/defocus depth and incident light intensity, printed images have various contour results. Process variation band (PV Band) computes the bitwise-XOR region between the outermost and innermost contour as shown in Figure 4.2a to evaluate the printing robustness.

Square L_2 Error: Given the target image $\vec{Z}_{target}$ and the printed image $\vec{Z}_{nominal}$, which represents the image printed via nominal lithography process condition, the square L_2 loss is calculated as $||\vec{Z}_{nominal} - \vec{Z}_{target}||_2^2$.

Edge placement error (EPE): Edge placement error is used to evaluate the difference of the contour between the target design $\vec{Z}_t$ and the image $\vec{Z}_{nom}$. To calculate the EPE, a series of points are sampled along the contour of the target design, as shown

in Figure 4.2b. If the distance $D(x, y)$ between the target design to the printed image is larger than an EPE constraint th_{EPE}, the point (x, y) is labeled as a EPE violation.

$$\text{EPE\_Violation}(x, y) = \begin{cases} 1 & D(x, y) \geq th_{EPE} \\ 0 & D(x, y) < th_{EPE} \end{cases} \tag{4.4}$$

Mask Manufacturing Shot Count: Since ILT naturally generates purely curvilinear features, conventional fracturing methods require a large number of small rectangles to approximate the shape. Mask Data Preparation (MDP) is used to fracture the shapes on the masks into non-overlapping rectangles, known as Variable Shaped-Beam (VSB) shots, to ensure mask printability. The shot count is used to evaluate the complexity of mask patterns.

With the evaluation metrics defined above, we formulate the mask optimization problem as follows:

Problem 1 (Mask Optimization)**.** *Given a target image $\vec{Z}_t$, the objective of mask optimization is to find a mask $\vec{M}$, whose printed image through the lithography process is supposed to be close to the target image and keep stable under different process conditions, such that the EPE, L_2 loss, PV Band and manufacturing shot count are minimized.*

4.2.3 Inverse Lithography Techniques

The objective of the conventional ILT-based method is to find an optimized mask $\vec{M}_{opt} = g^{-1}(\vec{Z}_t, \vec{C}_{nom})$, where $\vec{Z}_t$ is the design target, and $g(\cdot, \vec{C}_{nom})$ stands for the lithography process under the nominal process condition. Usually, we can not obtain the inverse function of g to compute the closed-form solution. The optimal mask is searched by computing the gradient of an objective function F_{obj} and using the gradient to guide the adjustment of each pixel value.

To calculate the gradient, all variables during the lithography process have to be continuous. Therefore, the binarized and constrained pixel values of mask $\vec{M}$ and printed image $\vec{Z}$ should be relaxed. To achieve this, we first introduce an auxiliary and unconstrained variable $\vec{P}$ and assume that $\vec{M}$ is determined by $\vec{P}$. The relationship between them is depicted by a sigmoid function in Equation (4.5).

$$\vec{M} = \frac{1}{1 + \exp\left(-\theta_M \vec{P}\right)}, \tag{4.5}$$

where θ_M defines the steepness of the sigmoid function.

Then the whole lithography process can be represented as: $\vec{P} \rightarrow \vec{M} \rightarrow \vec{I} \rightarrow \vec{Z}$. Note that the original function that maps $\vec{I}$ to $\vec{Z}$ is a threshold function as formulated in Equation (4.3), which is also undifferentiable. Therefore, we approximate it using another sigmoid function

$$\vec{Z} = \frac{1}{1 + \exp\left(-\theta_Z(\vec{I} - I_{th})\right)}, \tag{4.6}$$

where θ_Z defines the steepness and I_{th} represents the intensity threshold as shown in Equation (4.3). In this way, the whole process becomes differentiable and each iteration of the optimization algorithm can be formulated as follows:

$$\vec{P}^{(j)} = \vec{P}^{(j-1)} - \eta \frac{\partial F_{obj}}{\partial \vec{P}^{(j-1)}}, \tag{4.7}$$

where η is the step size of gradient descent. $\vec{P}^{(j)}$ indicates the variable $\vec{P}$ at the j-th iteration. After finally obtaining the $\vec{P}_{opt}$ by minimizing the objective function F_{obj}, we binarize $\vec{P}_{opt}$ to $\vec{M}_{opt}$, which is the final optimized mask solution.

4.3 The L2O-ILT algorithm

In this section, we first discuss an optimization mechanism in Section 4.3.1 called alternating optimization, which solves the issue that the conventional ILT [17] does not achieve satisfactory joint optimization of design targets under different conditions. Then, we develop our learning model L2O-ILT in Section 4.3.2, where each layer is constructed based on our proposed ILT algorithm with alternating optimization, and the whole architecture is equipped with strong prior knowledge and highly interpretable. The model training and refinement strategy is explained in Section 4.3.3 and Section 4.3.4. A technique called adaptive solution space is proposed in Section 4.3.5 to help our model accelerate the convergence rate as well as keep the solution quality.

4.3.1 Alternating Optimization

As illustrated in Section 4.2.3, the general implementation of ILT-based methods is to first define an objective function of the mask, which is then optimized using numerical approach. Therefore, the quality of final solution is closely related to the definition of the objective function. Given an input $\vec{P}$, the classical pixel-based ILT [17] gives the objective function to be minimized as follows:

$$\begin{aligned} F_{target} =& L_{nominal} + L_{out} + L_{in} \\ =& ||\vec{Z}_{nominal} - \vec{Z}_{target}||_2^2 + ||\vec{Z}_{out} - \vec{Z}_{target}||_2^2 + ||\vec{Z}_{in} - \vec{Z}_{target}||_2^2, \end{aligned} \tag{4.8}$$

where $\vec{Z}_{nominal} = g(\vec{P}, \vec{C}_{nominal})$, $\vec{Z}_{out} = g(\vec{P}, \vec{C}_{out})$ and $\vec{Z}_{in} = g(\vec{P}, \vec{C}_{in})$. $\vec{C}_{out}$ and $\vec{C}_{in}$ stand for two extreme conditions, under which the outer-most and inner-most images will be printed.

Under the guidance of F_{target}, the printed images under different process condi-

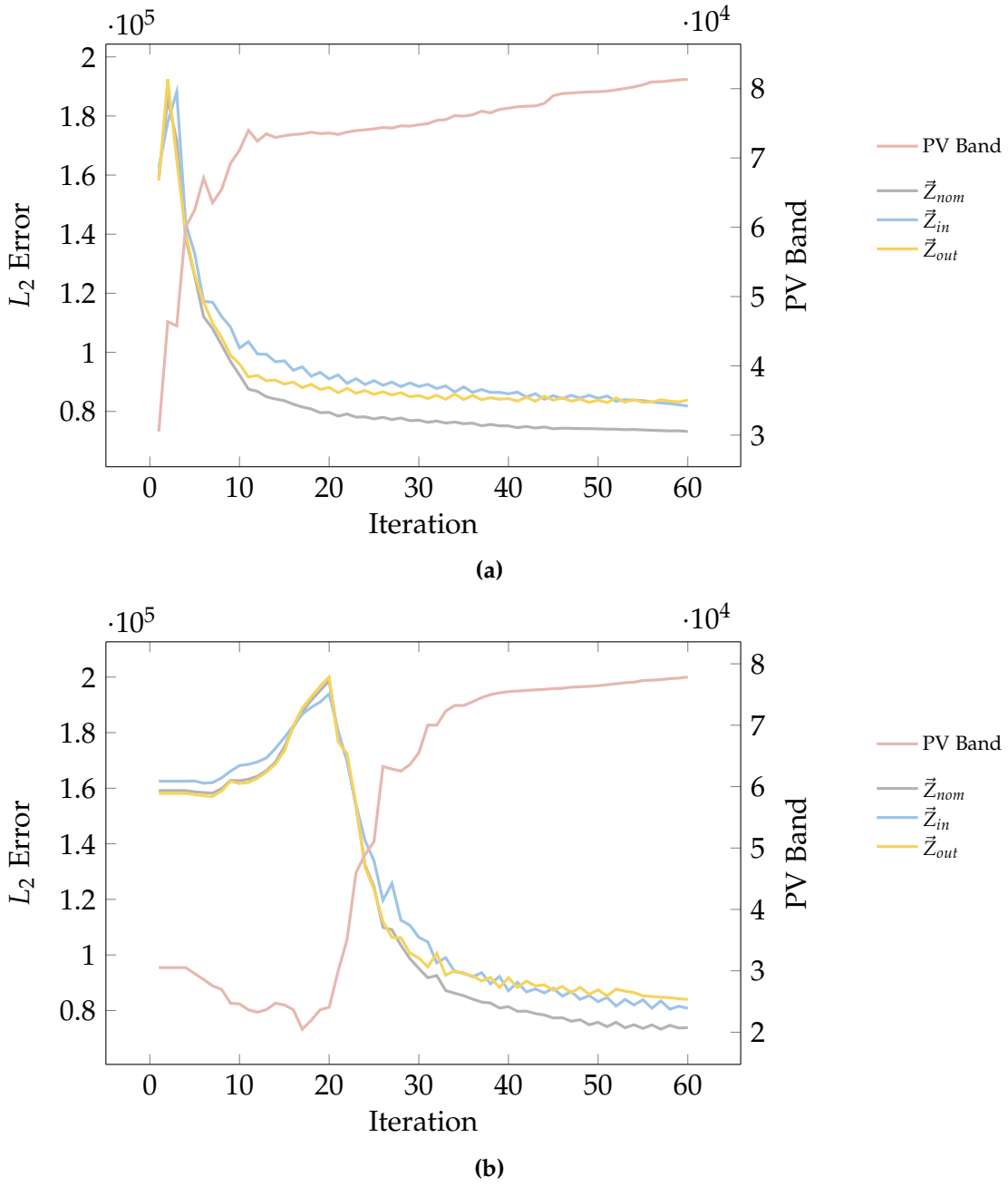


Figure 4.3: *The change of loss terms and PV Band in (a) conventional ILT; (b) the proposed alternating optimization scheme.*

tions are jointly pushed toward the target pattern, which is actually a desired property of an optimized mask. However, according to our empirical study

as shown in Figure 4.3a, we find that while all three loss terms are gradually minimized, the PV Band metric is negatively optimized. This is because minimizing $||\vec{Z}_{out} - \vec{Z}_{target}||_2^2 + ||\vec{Z}_{in} - \vec{Z}_{target}||_2^2$ can not guarantee the minimization of $||\vec{Z}_{out} - \vec{Z}_{in}||_2^2$ in mathematics. We also depict the change of $L_{nominal}$, L_{in}, L_{out}, and PV Band in Figure 4.3a. Therefore, although optimizing F_{target} contributes to reducing the error between the printed image and the real target, it results in a high PV Band value, leading to a large process window. We call such an objective function optimization "target-driven optimization" and we propose that the optimization configuration is supposed to be improved.

It can be easily seen that convergence rates of all three loss terms are drastically reduced after a certain number of iterations. Based on such an observation, we replace several iterations of optimizing F_{target} with optimizing F_{pvb}, formulated as:

$$F_{pvband} = ||\vec{Z}_{out} - \vec{Z}_{in}||_2^2, \tag{4.9}$$

which is directly related to PV Band performance. Optimizing F_{pvband} can be regarded as "PV Band-driven optimization". Instead of optimizing a weighted sum of F_{target} and F_{pvband}, we choose to decouple the optimization of these two objectives. This will make it more convenient to encapsulate basic modules in our deep learning-based framework, which will be shown in the following Section 4.3.2. And according to our experimental results shown in Figure 4.3b, we find that alternating "target-driven optimization" and "PV Band-driven optimization" can achieve satisfactory improvement in PV Band while slightly affecting $L_{nominal}$, L_{in}, and L_{out}. Our designed optimization scheme can be represented as:

$$\vec{P}^{(j)} = \begin{cases} \vec{P}^{(j-1)} - \eta \dfrac{\partial F_{pvband}}{\partial \vec{P}^{(j-1)}} & j < Q; \\ \vec{P}^{(j-1)} - \eta \dfrac{\partial F_{target}}{\partial \vec{P}^{(j-1)}} & j \geq Q, \end{cases} \tag{4.10}$$

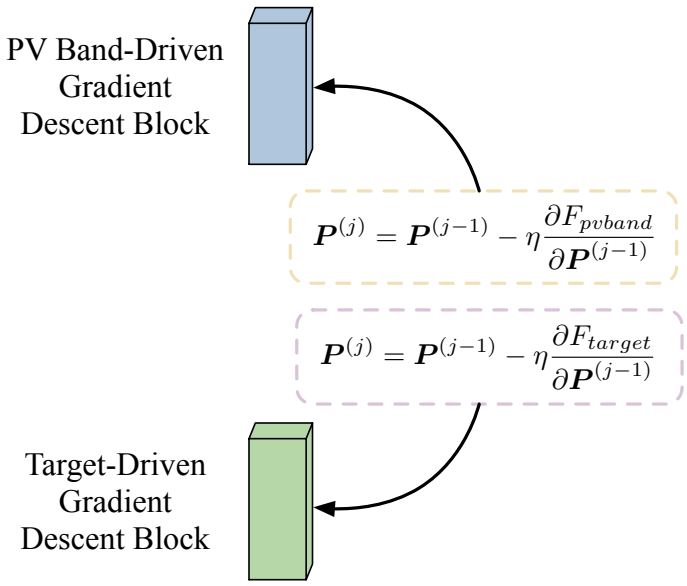


Figure 4.4: *Each iteration of L2O-ILT algorithm is represented as a neural network layer.*

where Q is a hyper-parameter to balance the performance of the process window and the L_2 error between the printed image and target design.

4.3.2 Model Architecture of L2O-ILT

Classic ILT-based mask optimization algorithms are built upon numerical approaches in a theoretically justified manner. In spite of the high interpretability, the performance heavily depends on human experiences, such as how to select appropriate parameters in the algorithm. Since these algorithms are sensitive to initial conditions and parameters chosen, the optimization results may be easily stuck in a local optimum state. Furthermore, a large number of optimization iterations are usually required to achieve an acceptable performance level, and thus these algorithms can be computationally expensive.

"Generative ILT" methods use learning-based models to quickly generate initial mask solutions and conduct further refinement. Regardless of its higher efficiency

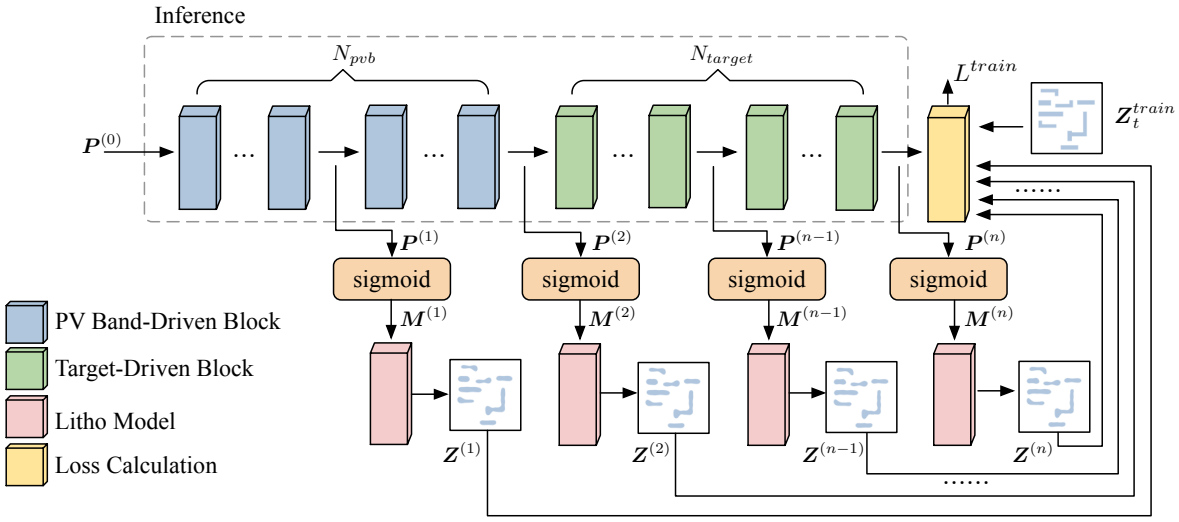


Figure 4.5: *Stacking multiple layers forms a neural network and passing through L2O-ILT is equivalent to executing an iterative ILT algorithm. Training L2O-ILT can be interpreted as tuning the parameter that is manually determined in the original ILT algorithm.*

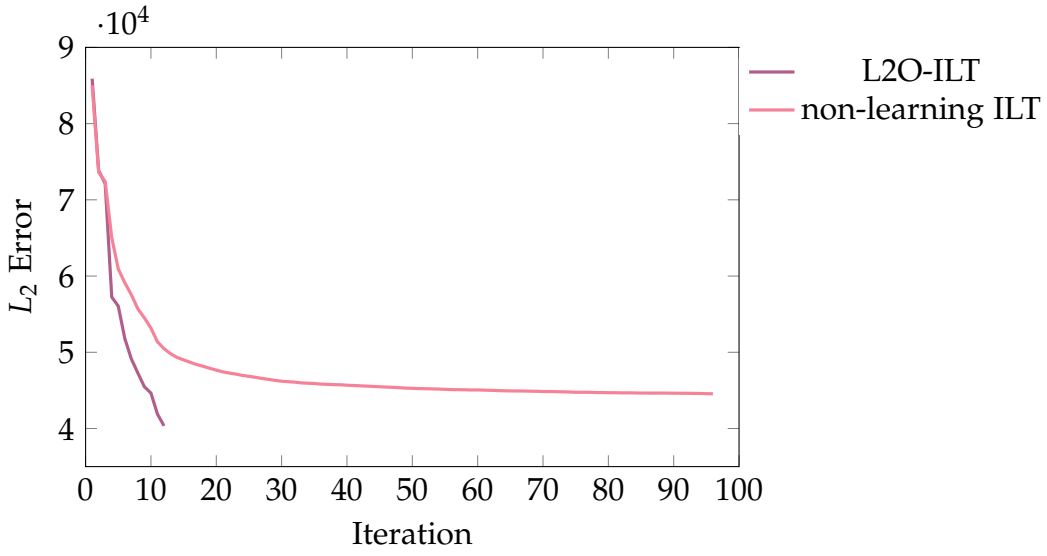


Figure 4.6: *Convergence rate comparison between conventional ILT and L2O-ILT model during inference.*

compared with conventional ILT, we notice that the quality of the initial mask is always low, which still requires a long-time correction to improve the solution quality. The inferior mask is mainly caused by the black-box property of generation

models, structures of which are difficult to be customized for mask optimization problems.

In accordance with the aforementioned observations, we propose an ILT algorithm-inspired learning model, L2O-ILT, which can generate a high-quality mask solution for fast refinement. The structure of L2O-ILT is not simply composed of stacking convolutional layers like "generative ILT" methods [18, 20, 19, 22]. Instead, each layer of our model is customized with prior knowledge for mask optimization tasks. To be specific, we unroll the entire ILT algorithm and use a neural network layer to represent each iteration of gradient descent as formulated in Equation (4.10), where the $\vec{P}^{(j)}$ and $\vec{P}^{(j-1)}$ can be regarded as the output and input of the j-th layer.

Based on our proposed alternating optimization scheme in Section 4.3.1, two kinds of neural network layers, target-driven block and PV Band-driven block, are respectively designed as shown in Figure 4.4. The architecture of L2O-ILT can be regarded as a time-unfolded recurrent neural network. In addition, our model can also keep the consistency advantage of ILT algorithms, i.e., we can still get a similar mask even if the pattern is offset. This is ensured by the translation invariance property of the lithography process as proved in [103].

The computation of the Target-Driven block exactly represents $\vec{P} = \vec{P} - \eta \frac{\partial F_{target}}{\partial \vec{P}}$. To further illustrate the concrete computation operation, we first represent the $\frac{\partial F_{target}}{\partial \vec{P}}$ as follows:

$$\frac{\partial F_{target}}{\partial \vec{P}} = \frac{\partial L_{nominal}}{\partial \vec{P}} + \frac{\partial L_{out}}{\partial \vec{P}} + \frac{\partial L_{in}}{\partial \vec{P}}. \tag{4.11}$$

The gradient calculations of all three terms are similar, and here we take $\frac{\partial L_{nominal}}{\partial \vec{P}}$ as

an example, which is computed by:

$$\begin{aligned}\frac{\partial L_{nominal}}{\partial \vec{P}} = 2\theta_Z\theta_M\vec{M}\circ(1-\vec{M})\{\vec{H}_{nominal}\otimes[(\vec{Z}_{nominal}- \\ \vec{Z}_{target})\circ\vec{Z}_{nominal}\circ(1-\vec{Z}_{nominal})\circ(\vec{M}\otimes\vec{H}^*_{nominal})]+ \\ \vec{H}^*_{nominal}\otimes[(\vec{Z}_{nominal}-\vec{Z}_t)\circ\vec{Z}_{nominal}\circ(1-\vec{Z}_{nominal})\circ \\ (\vec{M}\otimes\vec{H}_{nominal})]\},\end{aligned} \tag{4.12}$$

where "$\circ$" indicates the matrix element-wise multiplication and "$\otimes$" stands for the convolution operation. The PV Band-driven block is designed in a similar way, which precisely represents the computation of $\frac{\partial \vec{F}_{pvband}}{\partial \vec{P}}$. In addition, all convolution operations in our algorithm are implemented via FFT convolution to save computation resources. Since the optical kernel size is quite large, given a $k \times k$ (e.g., 35×35) optical kernel and $N \times N$ mask (e.g., 2048×2048), the computation complexity of FFT convolution is $\mathcal{O}(N^2 \log N^2)$, less than the complexity $\mathcal{O}(k^2N^2)$ of direct convolution. All the matrix computations can be easily implemented with the deep learning toolkit, such as Pytorch [104], which provides matrix computing with strong acceleration implemented by CUDA kernel.

Stacking N_{pvband} PV Band-driven blocks and N_{target} target-driven blocks forms a deep neural network, which is exactly our L2O-ILT as shown in Figure 4.5, and passing through the entire neural network is equivalent to executing the ILT algorithm a number of iterations. In L2O-ILT, we set both N_{pvband} and N_{target} as 5. The convergence rate of our L2O-ILT can be boosted via model training. All learnable parameters, such as the step size of gradient descent, updated during the training process are all from the original ILT algorithm. Therefore, the model training can be naturally interpreted as a parameter auto-tuning process to achieve much faster convergence than non-learning ILT with hand-crafted parameters. We show the convergence rate comparison in Figure 4.6. Compared with conventional

non-learning ILT, L2O-ILT is able to achieve convergence within a much smaller number of iterations, resulting in significant ILT acceleration. Passing through one layer can be regarded as executing the original non-learning ILT algorithms for multiple iterations. In addition, the learnable parameters can also help avoid the local optimum state, contributing to the robustness of our algorithm.

Note that the number of iterations also indicates the number of layers in L2O-ILT. The training strategy will be explained in Section 4.3.3. Overall, our proposed framework seamlessly incorporates the prior knowledge of mask optimization and achieves ILT acceleration with deep learning, and therefore we call it "learning to optimize ILT".

4.3.3 Interpretable Self-supervised Learning

In order to accelerate the convergence rate of L2O-ILT, a specialized interpretable training strategy is proposed. As illustrated in Section 4.3.2, each layer of our neural network is equivalent to an optimization iteration, and each layer outputs a mask that has not been fully optimized. Therefore, we can directly supervise the intermediate-generated mask $\vec{M}^{(i)}$ computed from $\vec{Z}^{(i)}$ using the training target design $\vec{Z}^{train}_{target}$. Such a training strategy can be regarded as providing a look-ahead mechanism, which forces $\vec{Z}^{(i)}$ to be close to the real target $\vec{Z}^{train}_{target}$. As a result, the error between the printed image $\vec{Z}^{(n)}$ of the final mask $\vec{M}^{(n)}$ and $\vec{Z}^{train}_{target}$ will be reduced efficiently. The training loss function can be formulated as:

$$L^{train} = \sum_{i=1}^{n} l^{(i)}(\vec{M}^{(i)}, \vec{Z}^{train}_{target}), \tag{4.13}$$

where n is a configurable hyper-parameter, representing the number of intermediate masks that we supervise with $\vec{Z}^{train}_{target}$, as shown in Figure 4.5. The loss between $\vec{M}^{(i)}$ and $\vec{Z}^{train}_{target}$ is decided by its printed image $\vec{Z}^{(i)}$, and the computation of $l^{(i)}$ is

calculated as:

$$l^{(i)} = ||\vec{Z}^{(i)}_{nominal} - \vec{Z}^{train}_{target}||_2^2 + ||\vec{Z}^{(i)}_{out} - \vec{Z}^{(i)}_{in}||_2^2, \tag{4.14}$$

where $\vec{Z}^{(i)}_{nominal}$, $\vec{Z}^{(i)}_{out}$ and $\vec{Z}^{(i)}_{in}$ stands for the printed image through our lithography module under different conditions. Such a training loss function design contributes to jointly optimizing PV Band, L_2 error and EPE. It should be reminded that **the parameters of all lithography modules mapping from $\vec{M}^{(i)}$ to $\vec{Z}^{(i)}$ are fixed and unlearnable**, so as to ensure the correctness of the lithography process.

In addition, it can be observed that our training scheme is self-supervised learning. To be specific, we directly adopt the target design as the supervision signal, which is totally different from previous "generative ILT" methods [19, 20, 22]. When given a set of training target design $\mathcal{Z}^{train}_{target}$, they demand a corresponding optimized mask set $\mathcal{M}^*$ acting as the "ground truth" signal to supervise the mask output by the black-box generation model. We argue that this training scheme is not reasonable. This is because when given a target design, there is no way to know what its actual corresponding mask is. Therefore, the optimized masks $\mathcal{M}^*$ utilized by previous methods [19, 20, 22] are actually approximated optimized masks, which are obtained from conventional ILT algorithms. In this way, the "ground truth" signals themselves are not accurate to act as the supervision signals for training. Moreover, the model training process will further accumulate the error. This also provides another reason to explain why the initial solution generated by these generation models is inferior.

4.3.4 Inference and Refinement

We have finished discussions about the model architecture and training process, both of which are highly interpretable. Overall, the model architecture is unrolling the

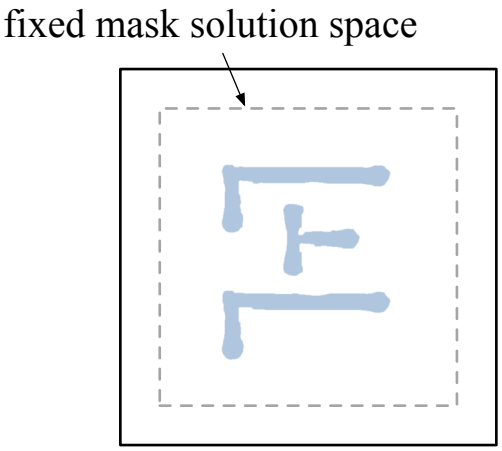


Figure 4.7: *The mask optimization result of a simple target design pattern.*

iterative algorithm, and the model training can be regarded as tuning the parameters to improve the original manual parameter configuration to accelerate optimization convergence.

Finally, when applied to design targets from the test dataset, our learned L2O-ILT model is able to generate a superior initial mask solution instantly. To further improve the mask quality, refinement is conducted on the initial mask. When given a test target $\vec{Z}_t^{test}$, the mask refinement is achieved by finetuning the parameters of the last layer by optimizing the following loss function:

$$L^{finetune} = \gamma||\vec{Z}^{(n)}_{nominal} - \vec{Z}^{test}_{target}||_2^2 + \frac{1}{\gamma}||\vec{Z}^{(n)}_{out} - \vec{Z}^{(n)}_{in}||_2^2, \tag{4.15}$$

where γ is an adaptive factor to balance these two loss terms, and it is computed as the ratio between the L_2 error and PV Band of the initial mask. Because of the high-quality initial mask, the refinement can be quickly converged within a very small number of iterations.

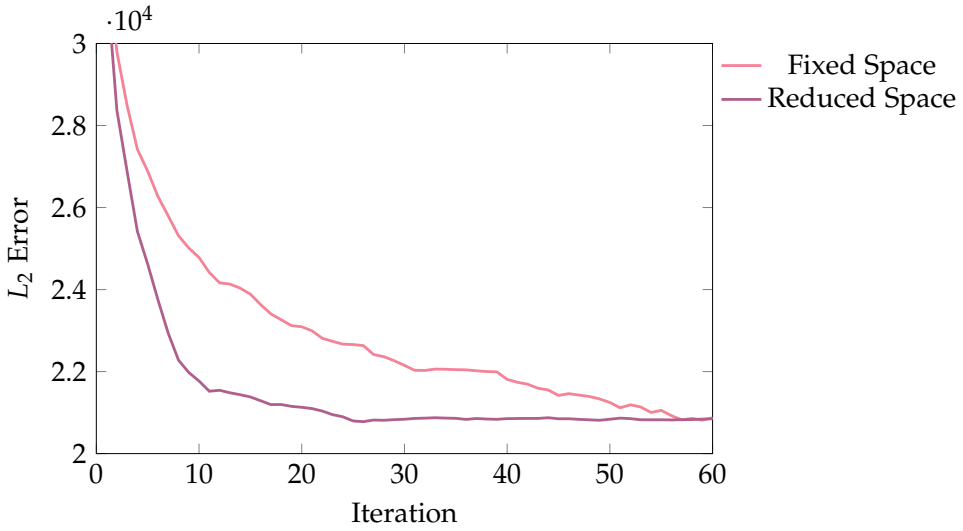


Figure 4.8: *Comparison of the convergence rate in fixed solution space and reduced solution space.*

4.3.5 Adaptive Solution Space

Conventional ILT algorithms [17, 18, 19, 20, 21, 22] optimize the mask pixel within a determined wide-ranging area, e.g., 1280×1280. However, we observe that the areas of the optimized mask are always close to the targets. Specifically, the updated pixels always lie in the neighbourhood of the target patterns. Therefore, we try to leverage this prior knowledge in our framework. We propose that the space can be suitably narrowed while keeping a high-quality mask solution. Based on such a motivation, we design an adaptive solution space in our algorithm, and this mechanism can also effectively avoid the emergence of outlier features in the optimized masks. In addition, a smaller solution space will also contribute to a faster convergence rate as well as saving computation resources. The experimental results show that the convergence rate of mask optimization will increase, as shown in Figure 4.8.

We design an adaptive mechanism to dynamically adjust the solution space according to the specific target patterns. It is based on such an observation that

the optimized mask area always lies in the neighborhood of the target pattern. Therefore, our adaptive solution space is achieved by expanding the target pattern. Usually, this can be implemented by convolution with a square kernel $\mathbf{1} \in \mathbb{R}^{s \times s}$, where $\mathbf{1}$ is a matrix in which all elements are 1, and N is the image size. Such a dilation operation is feasible but not efficient, which has $\mathcal{O}(s^2N^2)$ complexity. To solve this issue, an agile algorithm is designed to satisfy our requirements. To specific, we can directly move the vertices to adjust the solution space. It is noted that the layout patterns tested in this work are from ICCAD 2013 CAD Contest [102] where all patterns are regular polygon shapes and represented as a vector of vertices as shown in Figure 4.9. Therefore, all vertices are off the shelf. And there are no extra workloads to transfer the pixel representation to the vertex representation. The movement direction of each vertex $\vec{v}_i = (x_i, y_i)$ is determined by its convexity-concavity and two neighborhood vertices $\vec{v}_{i-1}$, $\vec{v}_{i+1}$, which can be formulated as:

$$\begin{aligned}\vec{u} &= (\vec{v}_{i+1} - \vec{v}_i) \times (\vec{v}_i - \vec{v}_{i-1}) \\ &= (0, 0, (x_{i+1} - x_i)(y_i - y_{i-1}) - (y_{i+1} - y_i)(x_i - x_{i-1})),\end{aligned} \tag{4.16}$$

$$c = \text{sign}(\vec{u}_z), \tag{4.17}$$

$$x_i' = x_i + c \cdot \text{sign}((x_i - x_{i-1}) - (x_{i+1} - x_i)) \cdot \text{offset}, \tag{4.18}$$

$$y_i' = y_i + c \cdot \text{sign}((y_i - y_{i-1}) - (y_{i+1} - y_i)) \cdot \text{offset}. \tag{4.19}$$

In Equation (4.16), to compute the cross product, we assume that vectors $(\vec{v}_{i+1} - \vec{v}_i)$ and $(\vec{v}_i - \vec{v}_{i-1})$ have a 0 z-axis component. The coefficient c is to determine whether the vertex is convex or concave according to the positive or negative of the z-axis component $\vec{u}_z$ of $\vec{u}$, and sign$(\cdot)$ represents the sign function. We have $c = 1$ when the vertex is convex. (x_i', y_i') indicates the coordinate of vertex $\vec{v}_i$ after movement,

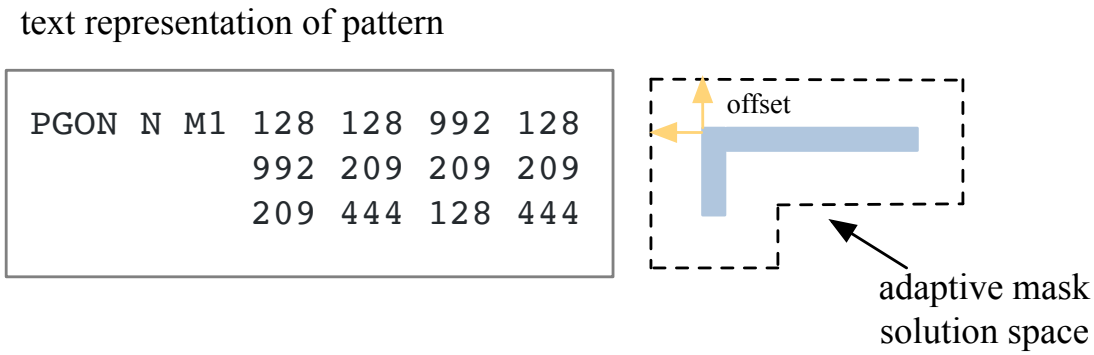


Figure 4.9: *Adaptive solution space via the movement of vertices.*

and "offset" is a configurable hyper-parameter to control the size of solution space and further accelerates the convergence rate.

Adjusting the space by moving vertices requires $\mathcal{O}(p)$ computation complexity, where p represents the number of vertices, typically less than 100. Therefore, such an algorithm is much more efficient than the traditional dilation operation with complexity $\mathcal{O}(s^2N^2)$. The generated adaptive solution space $\vec{S}_{ada}$ can be incorporated into our original gradient descent formulation in Figure 4.4 to restrict the range of mask pixels update. As shown in Figure 4.9, we only allow the pixels within the space to be updated during the optimization process. The gradient descent formulations combined with the adaptive solution space are now represented as:

$$\vec{P}^{(j)} = \vec{P}^{(j-1)} - \eta \frac{\partial F_{pvband}}{\partial \vec{P}^{(j-1)}} \circ \vec{S}_{ada}, \tag{4.20}$$

$$\vec{P}^{(j)} = \vec{P}^{(j-1)} - \eta \frac{\partial F_{target}}{\partial \vec{P}^{(j-1)}} \circ \vec{S}_{ada}, \tag{4.21}$$

where $\vec{S}_{ada}$ acts as a filter. In $\vec{S}_{ada}$, the values within the adaptive solution space are 1 and others are 0.

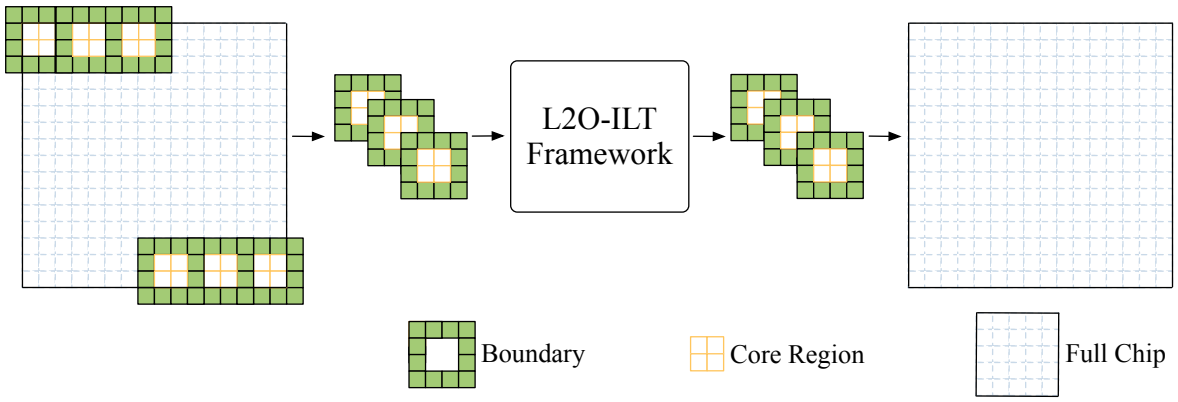


Figure 4.10: *The adaptation of L2O-ILT on full chip.*

4.3.6 Applied on Full Chip

The above methodology mainly discusses mask optimization on layout patterns of small size, i.e., 2048×2048. With the development of semiconductors and the shrinking size of transistors, the chip scale is constantly growing, which is usually much larger than the patterns used in academic research. To overcome this issue, we also explore the adaption of our L2O-ILT on the full chip. Inspired by [105], our proposed algorithm is illustrated in Figure 4.10, a combination of our L2O-ILT and the large tile global perception algorithm proposed by [105]. As shown in Figure 4.10, we adopt a sliding window to scan over the entire chip, dividing the large full chip into smaller chips. It is noted that each window includes two parts, the core region and the boundary region. The layout patterns within the boundary region in each sliding window will be ignored, and the core part is the mask region that we want to obtain its mask optimization result. As discussed in [105], such a sliding-window manner can help minimize boundary distortion effects. After feeding each tile into our L2O-ILT framework, the optimized mask of all core parts can be obtained, which will then be concatenated back. The stitching result is the optimized mask of the full chip.

Table 4.1: *Benchmark information of ICCAD 2013 Dataset*

Bench	Area (nm^2)
case1	215344
case2	169280
case3	213504
case4	82560
case5	281958
case6	286234
case7	229149
case8	128544
case9	317581
case10	102400

Table 4.2: *Mask printability and runtime comparison with state-of-the-art methods (Experimental settings follow [17])*

Bench	ILT [17]				GAN-OPC [19]				DevelSet [22]				L2O-ILT			
	EPE	L_2 (nm^2)	PVB (nm^2)	TAT (s)	EPE	L_2 (nm^2)	PVB (nm^2)	TAT (s)	EPE	L_2 (nm^2)	PVB (nm^2)	TAT (s)	EPE	L_2 (nm^2)	PVB (nm^2)	TAT (s)
case1	6	49893	65534	318	8	52570	56267	358	10	49142	59607	1.50	3	39742	50432	0.73
case2	10	50369	48230	256	13	42253	50822	368	1	34489	52010	1.40	0	31550	42620	0.72
case3	59	81007	108608	321	51	83663	94498	368	64	93498	76558	1.29	22	67612	73850	0.76
case4	1	20044	28285	322	2	19965	28957	377	2	18682	29047	1.65	1	12550	20306	0.72
case5	6	44656	58835	315	8	44733	59328	369	1	44256	58085	0.91	0	34056	50982	0.72
case6	1	57375	48739	314	12	46062	52845	364	2	41730	53410	0.84	0	31830	47237	0.73
case7	0	37221	43490	239	7	26438	47981	377	0	25797	46606	0.76	0	20443	37207	0.75
case8	2	19782	22846	258	0	17690	23564	383	0	15460	24836	1.14	0	13429	19702	0.74
case9	6	55399	66331	322	12	56125	65417	383	0	50834	64950	1.21	0	39652	58708	0.72
case10	0	24381	18097	231	0	9990	19893	366	0	10140	21619	0.42	0	8363	17561	0.71
Average	9.10	44012.70	50899.50	289.60	11.30	39948.90	49957.20	371.30	8.00	38402.80	48672.80	1.11	**2.60**	**29922.70**	**41860.50**	**0.73**
Ratio	3.500	1.471	1.216	396.712	4.346	1.335	1.193	508.630	3.077	1.283	1.163	1.523	**1.000**	**1.000**	**1.000**	**1.000**

4.4 Experimental Results

We implement our entire framework L2O-ILT with the Pytorch library [104] and test it on a Linux system with 2.3GHz Intel Xeon CPU and a single Nvidia GeForce RTX

Table 4.3: *Mask printability and runtime comparison with state-of-the-art methods (Experimental settings follow [20])*

Bench	Neural-ILT [20]				A2-ILT [106]				L2O-ILT			
	EPE	L_2 (nm^2)	PVB (nm^2)	TAT (s)	EPE	L_2 (nm^2)	PVB (nm^2)	TAT (s)	EPE	L_2 (nm^2)	PVB (nm^2)	TAT (s)
case1	8	49817	55975	13.96	7	45824	59136	4.43	3	39636	46905	1.12
case2	3	38174	52010	15.87	3	33976	52054	4.48	0	29108	37099	1.11
case3	52	89411	91357	12.95	62	94634	82661	4.52	21	67263	69115	1.13
case4	2	16744	29982	9.53	2	20405	29435	4.44	1	10807	20694	1.12
case5	3	45598	58900	8.43	1	37038	62068	4.47	0	31909	48797	1.10
case6	5	43836	54969	8.50	2	40701	54842	4.44	0	31474	45453	1.14
case7	0	20324	50542	13.09	0	21840	48474	4.42	0	16942	35942	1.11
case8	0	13337	26353	12.94	0	14912	24598	4.47	0	12236	19496	1.13
case9	2	49401	68817	12.95	2	47489	68056	4.50	0	34849	56706	1.11
case10	0	8511	20734	11.66	0	9399	20243	4.35	0	7203	15976	1.11
Average	7.50	37515.30	50963.90	11.99	7.90	36621.80	50156.70	4.45	**2.50**	**28142.70**	**39618.30**	**1.12**
Ratio	3.000	1.333	1.286	10.705	3.160	1.301	1.266	4.045	**1.000**	**1.000**	**1.000**	**1.000**

3090 GPU. The evaluation data to test the model performance are from ICCAD 2013 CAD Contest [102], which includes ten industrial M1 designs on the 32nm design node and also provides the lithography engine. The dataset used for training our L2O-ILT is obtained from the authors of GAN-OPC [19].

4.4.1 Comparison with State-of-the-art

We compare the performance of the proposed L2O-ILT with other state-of-the-art mask optimization methods, and the detailed results are listed in Table 4.2 and Table 4.3. We learned that there exists offsets between the initial generated mask between the work in [17], [19] and [22] and [20], [106] and we noted that the offset of the initial mask would slightly affect the mask printability and complexity. To make a fair comparison, we set two versions of L2O-ILT results following corresponding experimental settings.

As listed in Table 4.2, compared with classical ILT [17] (denoted as ILT), the L_2

and PV Band are reduced by 47.1% and 21.6% respectively, and the EPE count is less than one-third of [17]. Compared with two "generative ILT" GAN-OPC [19], and DevelSet [22], which respectively adopt GAN [30] and U-Net [32] to generate initial mask solution, our model L2O-ILT also shows superiority. The performance of L_2 achieves 33.5% and 28.3% enhancements, and PV Band could obtain 19.3% and 16.3% improvements. For the EPE count, the number of our EPE is only 2.60 on average, which is much smaller than GAN-OPC [19] (11.30) and DevelSet [22] (8.00). As for the runtime, our model is also faster than prior work. According to Table 4.2, compared with ILT [17], GAN-OPC [19] and DevelSet [22], our L2O-ILT achieves 396.712×, 508.630× and 1.523× speedup respectively.

As for the other experimental results listed in Table 4.3, when following the same settings as [20], L2O-ILT also achieves the best performance. Specifically, our model averagely outperforms Neural-ILT [20] with 33.3% and 28.6% reduction in L_2 error and PV Band. And the EPE count is only one-third of Neural-ILT [20]. Compared with A2-ILT [106], which relies on the reinforcement learning technique to improve the ILT performance, the L_2 and PV Band of our model are still reduced by around 30.1% and 26.6%. Also, our average EPE count is 2.50, much smaller than the EPE count of A2-ILT [106]. For the total runtime, L2O-ILT achieves 11.091× and 4.038× speedup in comparison with Neural-ILT [20] and A2-ILT [106].

Besides, we also evaluate the mask manufacturability in terms of the shot count, which stands for the number of rectangles that are used to approximate the optimized mask patterns. The comparison results are listed in Figure 4.11a and Figure 4.11b. Among the above-mentioned methods, the shot number of L2O-ILT is reduced by 99.2%, 8.3% and 7.2% compared with ILT [17], GAN-OPC [19] and DevelSet [22]. Although the masks generated by L2O-ILT contain 21.7% and 28.8% more shots than Neural-ILT [20] and A2-ILT [106], the mask printability and

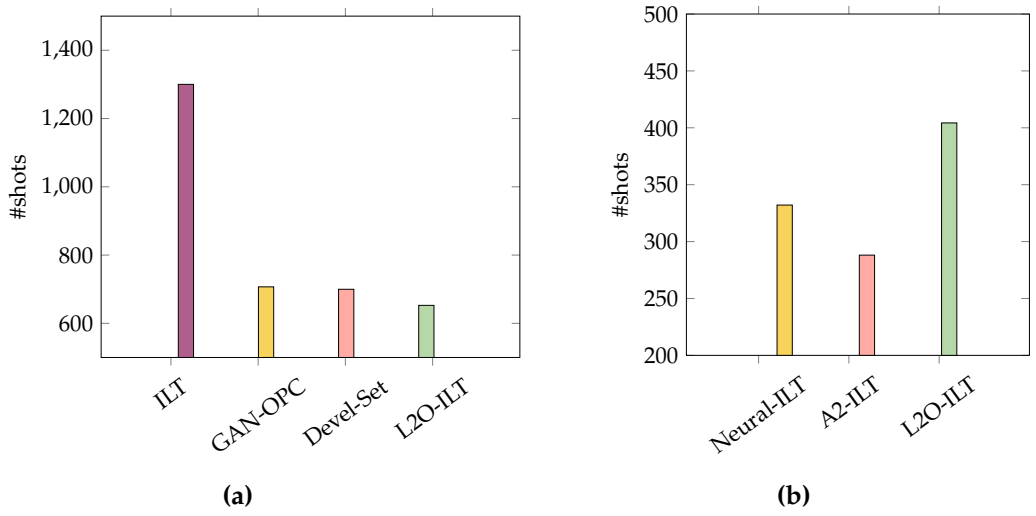


Figure 4.11: *Comparison of the mask manufacturability with state-of-the-art methods.*

runtime performance is much better as listed in Table 4.5. The quality and simplicity of the mask make a trade-off, and we think it is acceptable to remarkably improve the mask printability within less runtime at the cost of a little bit higher complexity.

In addition, the memory usage of L2O-ILT is also compared versus other state-of-art methods, and the comparison results are listed in Table 4.4. It can be seen that our proposed model requires 7.4GB GPU memory, which is comparable with other state-of-the-art methods. This also indicates that we successfully incorporate the prior knowledge of ILT into the deep learning-based model while not remarkably increasing the complexity of the model.

4.4.2 Evaluation of Initial Mask Qualities

To prove the benefit of our model that high-quality mask solutions can be generated by L2O-ILT, we compare the L_2 error of the initial solutions with other "generative ILT" methods. Also, considering different experimental settings, we split the result comparison into two groups, as shown in Figure 4.12a and Figure 4.12b. The average

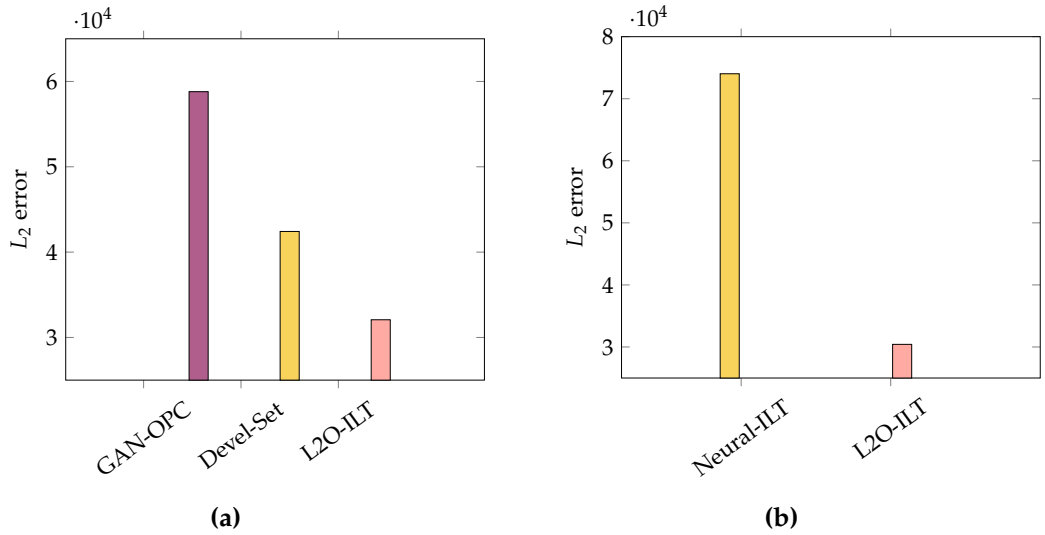


Figure 4.12: *Comparison of the initial mask solution with GAN-OPC [19] and DevelSet [22], and with Neural-ILT [20].*

L_2 error of our initial masks is much lower than the initial masks of GAN-OPC [19], DevelSet [22], and Neural-ILT [20]. (A2-ILT [106] is not considered as "generative ILT" since it does not adopt a generation model). Combined with the results in Table 4.2 and Table 4.3, we can observe that for these "generative ILT" methods, there exists a large gap between the performance of the initial mask and the final result. Therefore, a long-time refinement is always required. As for the initial solutions of L2O-ILT, the performance gap is really small, and thus the refinement can be finished rapidly, i.e., within 20 iterations, which contributes to the remarkable runtime improvement.

4.4.3 L2O-ILT Acts as a Plugin

Another benefit of L2O-ILT is that our model can be incorporated into other models like Neural-ILT [20] and GAN-OPC [19] as a plugin, which can improve their mask optimization performance. We take the Neural-ILT [20] as an example. Given an

Table 4.4: *Memory usage comparison with state-of-the-art methods*

	Memory Usage(GB)
GAN-OPC [20]	6.5
DevelSet [22]	8.0
Neural-ILT [20]	6.6
A2-ILT [106]	5.7
L2O-ILT	7.4

initial mask solution generated by Neural-ILT [20], the original refinement process in Neural-ILT [20] has to finetune the entire model, including the U-Net [32], which contains a lot of parameters. Therefore, the refinement process is not efficient.

An improved method is to combine L2O-ILT with Neural-ILT [20] by directly feeding the low-quality mask into our model. As explained in Section 4.3.4, given an initial solution and a test target pattern, the refinement process is achieved by tuning the last layer of our model, which includes fewer parameters than Neural-ILT [20]. Therefore, the refinement process is much more efficient to execute.

To verify the effectiveness of L2O-ILT as a plugin, we conduct further experiments using the ICCAD 2013 CAD benchmark [102]. We list the average performance of the mask refined by our L2O-ILT along with the required runtime in Table 4.5, where we use Neural-L2O-ILT to denote the combination of Neural-ILT [20] and L2O-ILT. It can be seen that in spite of the low-quality initial mask, with the L2O-ILT, the mask generated by Neural-ILT [20] can still be more efficiently refined and even achieve better results in comparison with original Neural-ILT [20]. Note that the "TAT" of Neural-L2O-ILT has considered the generation runtime of the initial mask.

Table 4.5: *Neural-ILT vs. Neural-L2O-ILT (Neural-L2O-ILT is to adopt L2O-ILT to refine the initial mask generated by Neural-ILT)*

	EPE	$L_2(nm^2)$	PVBand(nm^2)	TAT(s)
Neural-ILT [20]	9.20	38567.90	50636.70	11.80
Neural-L2O-ILT	3.00	30412.70	39626.70	1.81

4.4.4 Evaluation on Full Chip

We also evaluate the performance of L2O-ILT on a large-scale chip of size $144\mu m^2$ using the pipeline illustrated in Section 4.3.6. In our experiment, the large-scale chip is divided into smaller chips of size 2048×2048 and the size of the core region is set as 1024×1024. Such a setting effectively reduces the distortion effect while achieving satisfactory runtime performance.

We compare the performance of our proposed L2O-ILT and Neural-ILT [20] method in terms of L_2 and PV Band metrics. When evaluating the performance of Neural-ILT on the full-chip, we directly replace L2O-ILT in the pipeline shown in Figure 4.10 with Neural-ILT. The presented results demonstrate that L2O-ILT achieves a reduction of 8.0% and 12.2% in L_2 error and PV Band, respectively. These results illustrate the benefit of L2O-ILT for full-chip mask optimization.

4.5 Summary

In this chapter, we present L2O-ILT, a deep learning based-model that achieves mask optimization acceleration and keeps remarkable printability performance. Our model structure is implemented by unrolling our ILT algorithm, and thus the model structure is highly incorporated into prior knowledge of mask optimization. Such an ILT algorithm-inspired model is able to generate an initial mask solution with

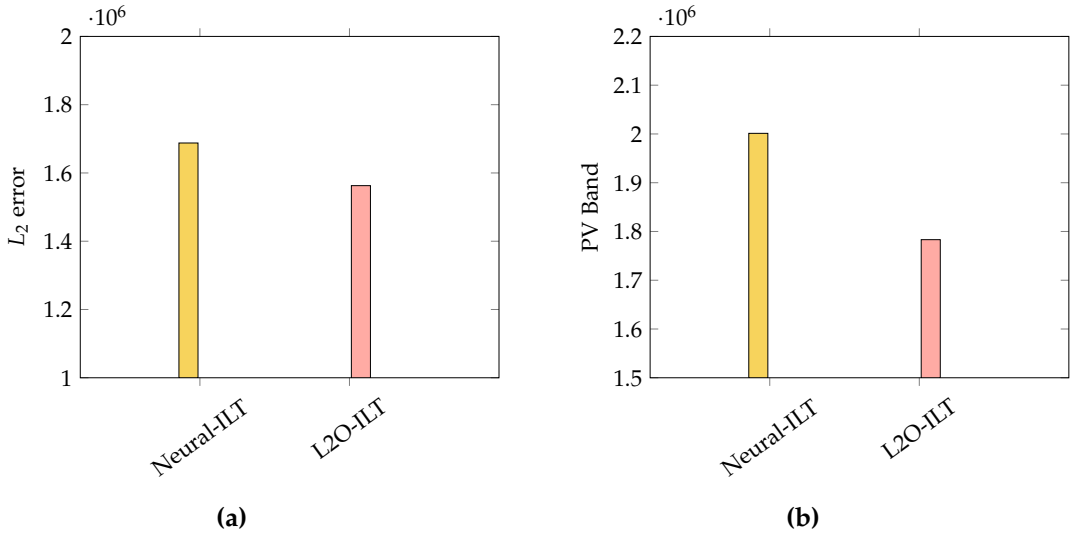


Figure 4.13: *Comparison of the full-chip mask optimization performance with Neural-ILT [20].*

better performance than previous methods, and the high-quality initial mask can be instantly refined to obtain the final solution. The experimental results demonstrate the superiority of our framework over current ILT acceleration works on both accuracy and efficiency.

Chapter 5

Hotspot Detection via Multi-task Learning and Transformer Encoder

5.1 Introduction

As semiconductor technology develops rapidly, the size of integrated circuit components is becoming much smaller. This poses a challenge for chip manufacturers since it is much more difficult to ensure the printability of layout designs due to shrinking feature sizes. Therefore, a precise and efficient hotspot detection technique is crucial to help locate the defect position of a given layout.

With the development of deep learning, learning-based hotspot detection methods [107, 108, 109, 110, 111, 112] show strong generalization abilities and achieve acceptable performance. Especially the approaches [113, 56, 57, 58, 114, 59, 115] built upon deep learning techniques get significant improvements on both accuracy and efficiency. However, there are still some issues with previous methods. (1) Most deep learning-based frameworks are designed to detect whether there is a hotspot at the center of an input clip. It would take a long time to detect the whole

layout design. Although the method proposed in [59] can detect multiple defects in large scales, some redundant designs may affect the efficiency. To be specific, the whole framework is a two-stage detector that requires a region proposal network as the first stage to select the potential defect regions. A refinement module is designed to process these candidate defect regions for more accurate results (2) Chen *et al*. [59] trains a framework to locate hotspots with rectangle bounding boxes. However, the underlying information, e.g., corner information, is not utilized as supervision for model training. The underlying information is helpful for detection tasks in some aspects. For example, corner information contributes to improve the localization accuracy [116], while center information performs better on detecting small targets [117]. (3) We observe some outlier situations where two regions sharing the same layout patterns may have different simulation results, i.e., one region is identified as a hotspot while the other is not. An example is shown in Figure 5.1a and Figure 5.1b. This phenomenon indicates that we can not judge whether a hotspot exists only by focusing on the local layout patterns, instead, context information plays an important role. However, CNNs, commonly adopted by previous methods, are infeasible to capture the long-range dependencies due to the locality inductive bias. Only neighboring pixels are taken into consideration when performing convolution operation, leading to a limited receptive field for each position, as illustrated in Figure 5.1c. Therefore, frameworks solely based on CNNs have limitations on hotspot detection.

In this chapter, we propose a single-stage detection framework, where we design two modules called center head and corner head to learn the underlying representations of the hotspots. These two modules act as auxiliary to help produce more accurate bounding boxes, which are used to indicate hotspot regions. A visualized example of different representations is shown in Figure 5.2. Our motivation is

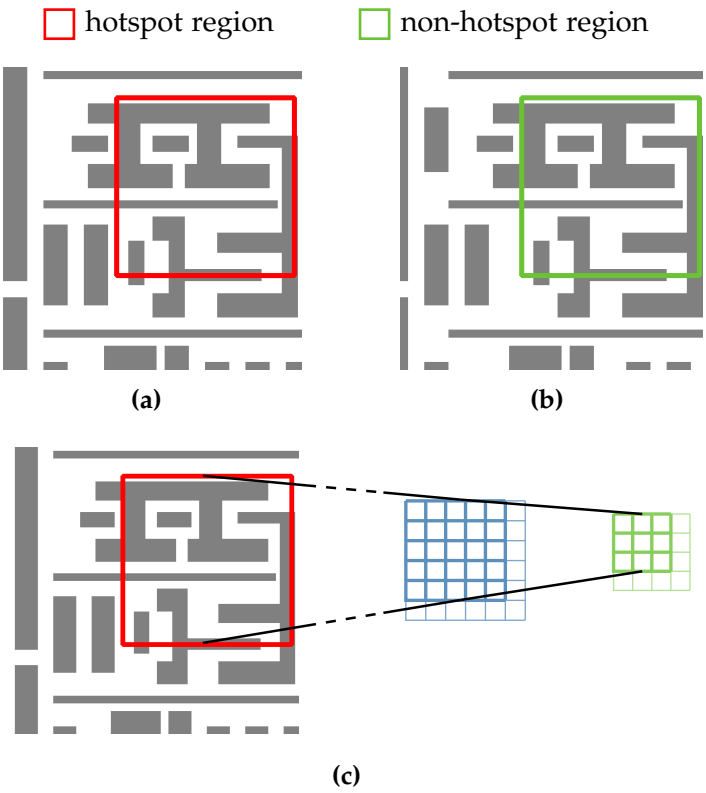


Figure 5.1: *The comparison between two regions with the same layout patterns. (a) Hotspot region. (b) Non-hotspot region. (c) Locality inductive bias of CNN.*

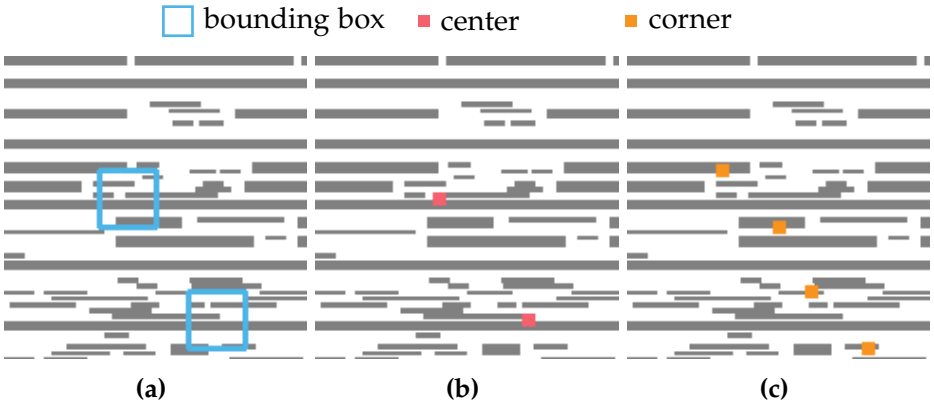


Figure 5.2: *The visualization for three different representations to indicate hotspot regions. (a) The bounding box of the hotspot region. (b) Center of the hotspot region. (c) Top-left and bottom-right corner of the hotspot region.*

that by jointly training the hotspot detector to learn different but related tasks, the knowledge learned from one task can be leveraged by others. This process simulates human perception and helps improve the overall performance. In addition, a feature aggregation module based on Transformer Encoder [118] is designed to augment the feature representation ability by modeling the dependencies between each feature with others. With this module, the issue of the CNNs, which can only attend to local features, is mitigated. The main contributions of this chapter are listed as follows:

- We propose a single-stage detector skipping the region proposal stage, which can effectively detect the hotspots.
- We build up center head and corner head to detect the center and corner points of the hotspot regions.
- We design a feature aggregation module and a feature sampling strategy to enrich the feature representation. The sampling strategy is adopted to save the computation cost of the aggregation operation.
- Experimental results show that our model achieves high detection accuracy and speed over prior state-of-the-art models.

The rest of this chapter is organized as follows. Section 5.2 introduces terminologies and evaluation metrics related to this work. Section 5.3 discusses each component of the proposed hotspot detector. Section 5.4 describes the implementation details including the loss function design. Section 5.5 shows the experimental comparison results with state-of-the-art, followed by the conclusion in Section 5.6.

5.2 Preliminaries

In this section, we will introduce the problem formulation and some preliminary knowledge related to this work.

5.2.1 Problem Formulation

In the process of chip manufacturing, lithography is employed to transfer layout patterns onto silicon wafers. However, this process involves many variations, and some patterns are sensitive to these variations, which may reduce the manufacturing yield due to potential open or short-circuit failures. Layout patterns that are sensitive to lithographic process variations are defined as *hotspots*.

A high-performance hotspot detector should correctly detect as many hotspots as possible and avoid mistaking non-hotspot patterns for hotspot patterns. The following metrics are adopted to evaluate the performance of a hotspot detector.

Definition 4 (Accuracy)**.** *The ratio between the number of correctly detected hotspots and the number of ground-truth hotspots.*

Definition 5 (False Alarm)**.** *The number of non-hotspot regions that are mistakenly identified as hotspot regions by the hotspot detector.*

Problem 2 (Hotspot Detection)**.** *The objective of hotspot detection is to train a detector on a collection of clips containing both hotspot and non-hotspot layout patterns. The detector aims to accurately locate and classify all hotspots and non-hotspots, to maximize detection accuracy while minimizing false alarms.*

5.3 Hotspot Detection Architecture

The proposed architecture overview of our framework is illustrated in Figure 5.3. (1) The first part is a backbone made up of ResNet-50 and Feature Pyramid Network. With the backbone, we can transfer the input clip to high dimensional features, and multiple level feature maps are generated for subsequent detection. (2) To conduct the hotspot detection task, we predefine dense regions on the output feature map of the backbone. The classification head performs label prediction to judge whether a region is a hotspot region or not. The localization head adjusts the predefined regions to fit the groundtruth hotspot regions better. (3) Corner head and center head receive the output from the backbone and identify the corners and centers of input clips. (4) The feature aggregation module is used to learn a rich hierarchy of associative features across different positions in the localization and classification head separately. The feature selection module selects informative features and filters out unimportant ones to save the computation cost.

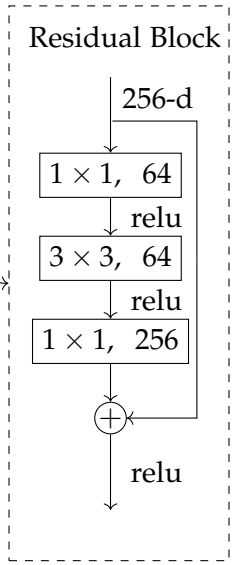
5.3.1 Backbone

Feature extraction is a process that identifies important features from input clips, which makes it easier for later hotspot detection. The deep convolutional neural network is a powerful tool for extracting effective features.

ResNet-50. Different from the previous works like [113, 59], our work does not concentrate on the design of the convolutional neural network for feature extraction. Instead, we adopt ResNet-50 [119] as the feature extractor, which proves to have prominent feature extraction ability according to much work like[119, 120, 121]. The architecture of ResNet-50 is listed in TABLE. 5.1.

Table 5.1: *ResNet-50 Architecture*

layer name	ResNet-50
conv1	7×7, 64, stride 2
conv2_x	3×3 max pool, stride 2
	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
conv3_x	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$
conv4_x	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$
conv5_x	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$



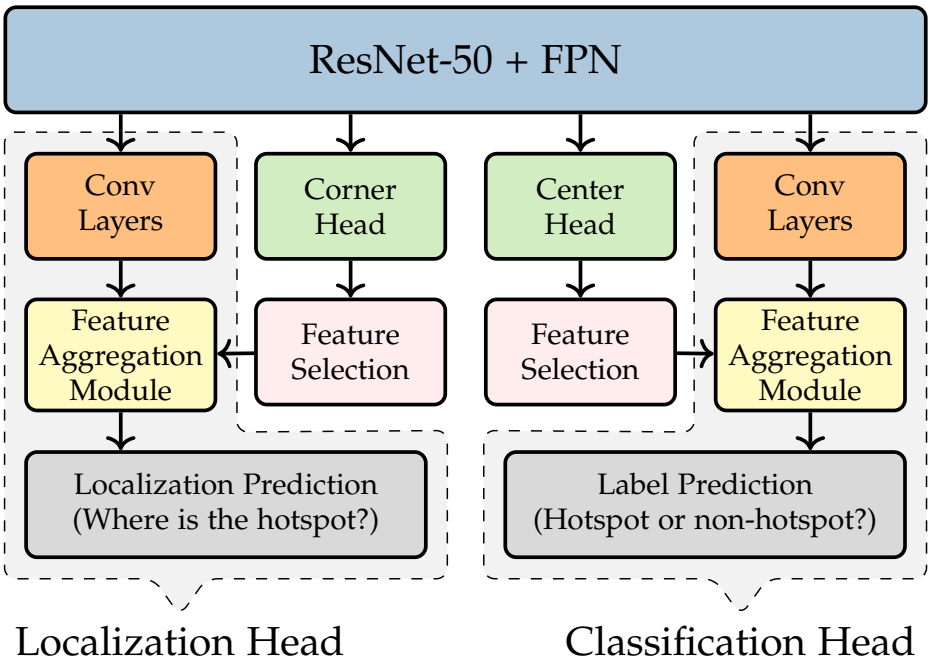


Figure 5.3: *The architecture of the proposed hotspot detector.*

Feature Pyramid Network. Previous work [113, 71, 59] simply detect hotspots on the output from the last layer of the backbone. However, we find that the feature maps of layers in different levels are also crucial for hotspot detection, which is further verified by our experimental results in 5.5.

To utilize the feature maps from different layers, we build the Feature Pyramid Network (FPN) [122] on top of the ResNet-50. As shown in the right part of Figure 5.4, with a top-down pathway and lateral connections, multi-scale feature maps are generated. Each feature map contains different level features, and all of them can be used for detecting hotspots. Considering the potential size of hotspot regions, we generate three feature maps (P_3, P_4, P_5) with different scales, where P_k indicates that it has resolution 2^k lower than the input.

5.3.2 Classification and Localization Head

In order to detect hotspots, previous work [59] first proposes the regions of interest (ROI) that are likely to contain the hotspot with region proposal network [59]. Then these proposed regions are fed into the detection head for regression and classification to get more accurate results. However, due to the region proposal network with complex structures, the detection speed is low.

To tackle this issue, we define dense regions which will be directly passed to classification and localization head after feature extraction. The region proposal step is skipped in order to improve the detection efficiency. These predefined regions are called anchors. More details about the anchor settings are illustrated in Section 5.4.1.

Assume that we define M anchors for each pixel of the feature map. Given a feature map $\vec{F} \in \mathbb{R}^{H \times W \times C}$ generated by the backbone as the input, the conv layers module, which consists of four consecutive 3×3 convolution layers with C filters, outputs a new feature map $\vec{F}_c \in \mathbb{R}^{H \times W \times C}$. A feature aggregation module is then adopted to enhance the representation ability of $\vec{F}_c$, and then produces another feature map $\vec{F}_a \in \mathbb{R}^{H \times W \times C}$. The structure of the feature aggregation module will be clearly illustrated in Section 5.3.4. The feature map F_a will be fed into the last layer, which is a 3×3 convolution layer with M filters and output the final result $\vec{F}_o \in \mathbb{R}^{H \times W \times M}$ where each element predicts the probability of each anchor containing a hotspot.

As for the regression head, it predicts the offset from each anchor to a nearby groundtruth hotspot, if one exists. The structure of the regression head is almost the same as the classification head except that the last layer has $4M$ filters. This is because the offset for each anchor is a 4-dimension vector, including the offset for a 2-d center, width, and height.

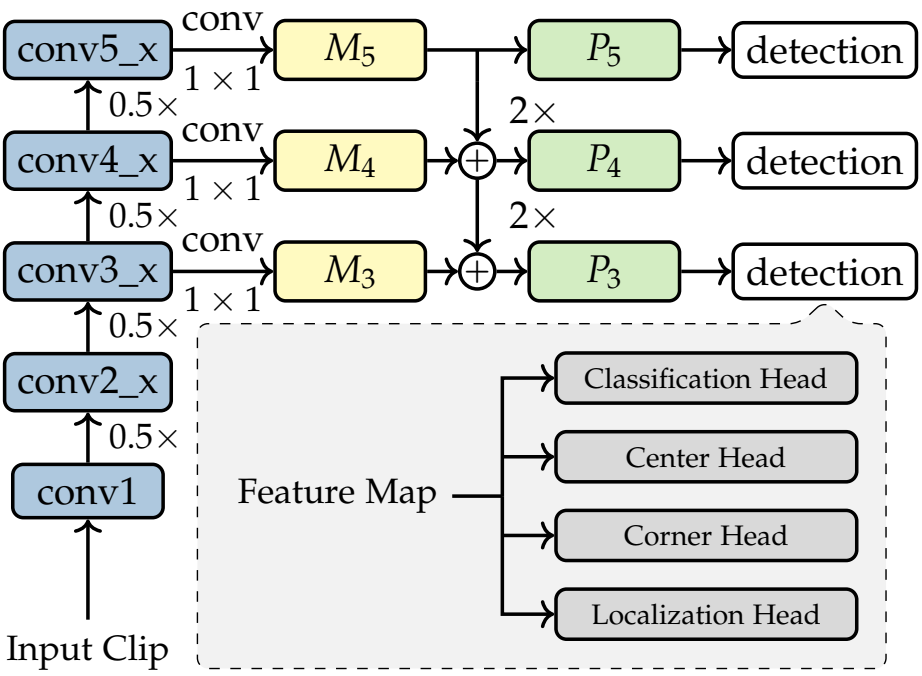


Figure 5.4: *The architecture of the backbone. The backbone is a combination of ResNet-50 and Feature Pyramid Network.*

5.3.3 Corner & Center Representation Learning

Multi-task learning is a learning paradigm which aims to learn multiple related tasks jointly so that the knowledge contained in one task can be leveraged by other tasks, with the hope of improving the overall performance.

In Section 5.3.2, the localization and classification heads are proposed to detect the potential hotspot regions with bounding boxes. Apart from the bounding box, corner and center representations are proposed in this work to further improve the detection performance. There are several advantages to utilize corner and center representation learning. (1) Center representation learning: hotspot region can be regarded as a small pattern in the layout design, for which center representation is proved to be friendly to detect small objects [117]. (2) Corner representation learning: corner representation is helpful for locating the target precisely [116].

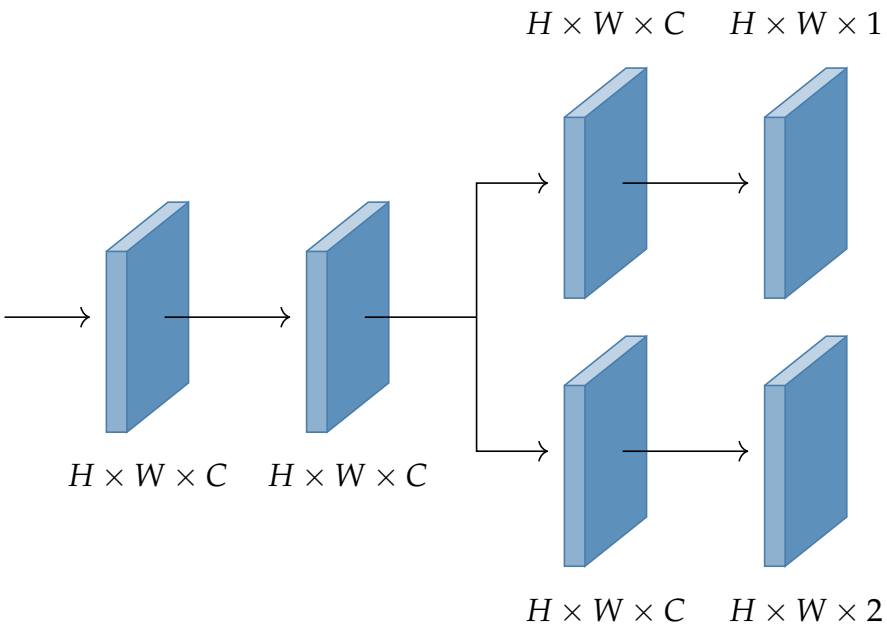


Figure 5.5: *Point head structure. The center head is composed of a single point head. The corner head is composed of two point heads.*

Following the paradigm of multi-task learning, corner and center representation learning share the same backbone with the previous heads illustrated in Section 5.3.2. To train the detector to learn these effective representations, we design corner head and center head separately. The center head receives the output $\vec{F} \in \mathbb{R}^{H \times W \times C}$ of the backbone and predicts the probability of each point being a center as well as the offset from each point to a nearby center, if one exists. The structure of the center head is composed of a single point head as shown in Figure 5.5. Different from the center head, the corner head identifies both the top left corner and bottom right corner of the hotspot region. Therefore, the corner head is composed of two separate point heads.

5.3.4 Feature Aggregation Module

The Transformer Encoder applied in machine translation tasks has shown its strength in modeling all pairwise interactions between different elements in a sequence. Inspired by its mechanism, we design a module called feature aggregation module (FAM) based on the Transformer Encoder.

With the help of FAM, we can enrich the feature map output from the conv layers module in the localization and classification heads, as shown in Figure 5.3. By globally capturing the dependencies between different features with the Multi-Head Attention mechanism, the representation ability of the feature map could be effectively augmented.

To be specific, given a feature map $\vec{F}_c \in \mathbb{R}^{H\times W\times C}$, when we hope to capture the dependencies between $\vec{f}_m \in \mathbb{R}^C$ with all other features $\vec{f}_n \in \mathbb{R}^C, n \in \{1, 2, \ldots, HW\}$, $\vec{f}_m$ is regarded as *query* and $\vec{f}_n$ is regarded as *key* and *value*. Noted that $\vec{F}_c$ is the output of the conv layers module as illustrated in Section 5.3.2. Then considering the operation via a single head of the Multi-Head Attention, which can be applied on $\vec{f}_m$ and $\vec{f}_n$ as follows:

$$\vec{h}_i = \sum_{n=1}^{HW} \frac{\exp(\vec{f}_m \vec{W}_i^Q (\vec{f}_n \vec{W}_i^K)^\top / \sqrt{C})}{\sum_{n=1}^{HW} \exp(\mathbf{f}_m \vec{W}_i^Q (\vec{f}_n \vec{W}_i^K)^\top / \sqrt{C})} \vec{f}_n \vec{W}_i^V, \tag{5.1}$$

where $\vec{h}_i \in \mathbb{R}^C$ is the output of head i. HW is the number of features of the feature map $\vec{F}_c \in \mathbb{R}^{H\times W\times C}$, and C is the dimension of the feature. $\vec{W}_i^Q, \vec{W}_i^K, \vec{W}_i^V \in \mathbb{R}^{C\times C}$ are projection matrices.

However, the computation cost is extremely expensive since the huge key set includes hundreds of candidates and the complexity for Equation (5.1) is $\mathcal{O}(HWC^2)$. In terms of this issue, we design a simple yet effective algorithm to reduce the amount of the selected keys. The main idea is to leverage the knowledge from the

center head and corner head to guide the key selection for the feature map in the classification head and localization head separately. The key selection algorithm on the center head is demonstrated in Algorithm 2. The key set of the corner head is selected in a similar way.

Algorithm 2 Key selection algorithm

Input: Center Probability Map $\vec{C} \in \mathbb{R}^{H \times W}$, Feature Map $\vec{F}_c \in \mathbb{R}^{H \times W \times C}$, Selection Number k;
Output: Key set $\vec{F}_k$;
1: $\vec{F}_k \leftarrow$ Initialized to empty set;
2: $\vec{C}' \leftarrow$ `AvgPool(`$\vec{C}$`)`;
3: $\vec{topk\_idx} \leftarrow$ the index of the maximum k values in $\vec{C}'$;
4: **for** $i \leftarrow 1, 2, \ldots k$ **do**
5: $\quad idx \leftarrow \vec{topk\_idx}[i]$;
6: $\quad \vec{f}_i \leftarrow$ the feature in the position idx of $\vec{F}_c$;
7: $\quad$ append feature f_i to key set $\vec{F}_k$;
8: **return** Key set $\vec{F}_k$ with k features.

As illustrated in Section 5.3.3, the probability map output by center head describes the probability of each point being a center. We perform a 3×3 average pooling with stride 1 on the probability map and get the positions of the maximum k values (lines 2–3). Then the features in the corresponding positions of the feature map are added to the key set (lines 4–8), which will be adopted to enhance the feature map.

Based on the new generated key set, we further design a variant of Equation (5.1) as follows:

$$\vec{h}_i = \vec{f}_m + \lambda \sum_{n=1}^{k} \frac{\exp(\vec{f}_m \vec{W}_i^Q (\vec{f}_n \vec{W}_i^K)^\top / \sqrt{C})}{\sum_{n=1}^{k} \exp(\vec{f}_m \vec{W}_i^Q (\vec{f}_n \vec{W}_i^K)^\top / \sqrt{C})} \vec{f}_n \vec{W}_i^V \tag{5.2}$$

where k is the selection number of keys defined in Algorithm 2 and $\vec{f}_n$ is from the key set $\vec{F}_k$. λ is a hyperparameter to control the feature augmentation degree. Compared to Equation (5.1), the complexity of Equation (5.2) is reduced to $\mathcal{O}(kC^2)$, where k

is much smaller than HW. In addition, the Multi-Head Attention is composed of multiple attention heads. In our framework, the number of the heads is set to 8.

5.4 Implementation Details

5.4.1 Anchors

For each pixel of the feature map, anchors with three different aspect ratios {1:2, 1:1, 2:1} and four different sizes $\{2^0, 2^{1/4}, 2^{2/4}, 2^{3/4}\}$ are set for dense scale coverage. Since each pixel is assigned with 12 anchors and each feature map is composed of many pixels, the number of anchors is extremely large, leading to the low efficiency for training. We can benefit from getting rid of large parts of the anchors. We first define Intersection-over-Union (IoU) as follows:

$$\text{IoU} = \frac{\text{anchor} \cap \text{groundtruth}}{\text{anchor} \cup \text{groundtruth}}. \tag{5.3}$$

Our assignment rule is based on the IoU between anchor and groundtruth as follows:

- If the IoU between an anchor and a groundtruth is larger than 0.5, the anchor will be regarded as positive sample.
- If the IoU between an anchor and any other groundtruth is smaller than 0.4, the anchor will be regarded as negative sample.
- If the IoU between a groundtruth and any other anchor is smaller than 0.5, the anchor with the highest IoU will be regarded as positive sample.
- The rest anchors are ignored during training.

Noted that each anchor is assigned to at most one groundtruth bounding box.

5.4.2 Training Loss

The objective function presented in this work is formulated as follows:

$$L^{det} = L^{bbox} + L^{ctr} + L^{cor}, \tag{5.4}$$

L^{bbox}, L^{ctr} and L^{cor} are bounding box loss, center loss and corner loss correspondingly. The detail explanation on these three terms will be introduced in this section.

Table 5.2: *Benchmark Information*

Bench	Train #HS	Test #HS	Train #Clips	Test #Clips	Training Set Size ($\mu m \times \mu m$)	Test Set Size ($\mu m \times \mu m$)
case2	40	39	1000	8	6.95 × 3.75	6.95 × 3.75
case3	1388	1433	1000	33	12.91 × 10.07	12.91 × 10.07
case4	90	72	1000	55	79.95 × 42.13	79.95 × 42.13

Table 5.3: *Comparison with State-of-the-art*

Bench	TCAD'19[71]			DAC'19[59]			Ours		
	Accu(%)	FA	Time(s)	Accu(%)	FA	Time(s)	Accu(%)	FA	Time(s)
case2	77.78	48	60.0	93.02	17	2.0	94.87	6	1.0
case3	91.20	263	265.0	94.5	34	10.0	97.2	26	4.0
case4	100.00	511	428.0	100.0	201	6.0	100	70	6.0
Average	89.66	274.00	251.00	95.84	84.00	6.00	**97.31**	**34.00**	**3.67**
Ratio	0.92	8.06	67.84	0.98	2.47	1.62	**1.00**	**1.00**	**1.00**

Bounding Box Loss: Based on the assignment rule in section 5.4.1, the ground truth classification target p_i is set to 1 if the anchor i belongs to positive sample and 0 otherwise.

Focal loss[123] is adopted to train the classification head. The classification loss

function is defined as:

$$L_{cls}^{bbox}(p_i, p_i') = \begin{cases} -\alpha(1-p_i')^{\gamma}\log p_i', & p_i = 1, \\ -(1-\alpha)p_i'^{\gamma}\log(1-p_i'), & p_i = 0, \end{cases} \tag{5.5}$$

where α and γ are hyperparameters. p_i' is the prediction result which indicates the probability containing the hotspot.

In addition to the classification head, the regression head predicts the offset between each anchor and its assigned groundtruth bounding box. $\vec{t}_i' = (t_x', t_y', t_w', t_h')$ and $\vec{t}_i = (t_x, t_y, t_w, t_h)$ are 4-d vector representing the regression prediction and the corresponding groundtruth target, respectively. They are defined as:

$$\begin{aligned} t_x &= (x - x_a)/w_a, \quad t_y = (y - y_a)/h_a, \\ t_w &= \log(w/w_a), \quad t_h = \log(h/h_a), \\ t_x' &= (x' - x_a)/w_a, \quad t_y' = (y' - y_a)/h_a, \\ t_w' &= \log(w'/w_a), \quad t_h' = \log(h'/h_a), \end{aligned} \tag{5.6}$$

where (x, y, w, h) represents the center coordinates, width and height. w', w_a and w represents the width of the predicted bounding box, anchor and groundtruth bounding box separately (same as x, y and h). Smooth L_1 loss is adopted for regression loss function which is formulated as:

$$L_{reg}^{bbox}(\vec{t}_i, \vec{t}_i') = \sum_{j=1}^{4} l_{reg}^{bbox}(t_i[j], t_i'[j]), \tag{5.7}$$

where

$$l_{reg}^{bbox}\left(t_i[j], t_i'[j]\right) = \begin{cases} \frac{1}{2}\left(t_i[j] - t_i'[j]\right)^2, & \text{if } \left|t_i[j] - t_i'[j]\right| < 1 \\ \left|t_i[j] - t_i'[j]\right| - \frac{1}{2}, & \text{otherwise.} \end{cases} \tag{5.8}$$

With the defined regression and classification loss function, the overall loss for the bounding box is calculated as:

$$L^{bbox} = \frac{1}{N_{anch}} \sum_i (L_{cls}^{bbox}(p_i, p_i') + p_i L_{reg}^{bbox}(\vec{t_i}, \vec{t_i'})), \tag{5.9}$$

where N_{anch} is the number of anchors.

Center & Corner Loss: Center and corner representations are adopted as the auxiliary to enhance the bounding box representation. To learn the effective feature expression, loss functions are designed for the corner head and the center head respectively. The loss for each point in the corner head is designed identically as the loss for the center head, thus we take the loss function for the center head as an example.

Similar to the anchor assignment, all the points within a feature map are divided into two categories. The points within the groundtruth bounding box are regarded as positive points and the rest are regarded as negative points. We assign each positive point with its corresponding bounding box center. For negative points, the groundtruth probability q_{xy} for (x, y) being the center is set to 0. And for positive points, we take the Gaussian kernel to describe the probability as follows:

$$q_{xy} = \exp\left(-\frac{(x - \lfloor\hat{x}\rfloor)^2 + (y - \lfloor\hat{y}\rfloor)^2}{2\sigma^2}\right), \tag{5.10}$$

where $(\hat{x}, \hat{y})$ is the assigned center for each positive point and σ is a hyperparameter. The classification branch of the center head predicts the probability q_{xy}' of each point being the center. A variant of focal loss [123] is adopted for the classification loss

function, defined as:

$$L_{cls}^{ctr}(q_{xy}, q'_{xy}) = \begin{cases} \left(1 - q'_{xy}\right)^{\theta} \log\left(q'_{xy}\right), & q_{xy} = 1, \\ \left(1 - q'_{xy}\right)^{\beta} \left(q'_{xy}\right)^{\theta} \log(1 - q'_{xy}), & q_{xy} < 1, \end{cases} \tag{5.11}$$

where θ and β are hyperparameters.

Besides predicting the probability of each point being a center, the center head is trained to predict the offset $\vec{d}'_{xy}$ between each positive point (x, y) and its corresponding center $(\hat{x}, \hat{y})$. Similar to eq. (5.7), the regression loss function is defined as:

$$L_{reg}^{ctr}(\vec{d}_{xy}, \vec{d}'_{xy}) = \sum_{j=1}^{2} l_{reg}^{ctr}(d_{xy}[j], d'_{xy}[j]), \tag{5.12}$$

where l_{reg}^{ctr} is taken as the same form as eq. (5.8).

By combining the regression and classification loss functions, the overall loss for the corner head is calculated as:

$$L^{ctr} = \frac{1}{N_{ctr}} \sum_{x} \sum_{y} (L_{cls}^{ctr}(q_{xy}, q'_{xy}) + \mathbb{I}_{q_{xy}>0} L_{reg}^{ctr}(\vec{d}_{xy}, \vec{d}'_{xy})), \tag{5.13}$$

where N_{ctr} is the number of centers for a given input. The indicator function $\mathbb{I}_{q_{xy}>0}$ points out that the negative points are omitted for regression part.

5.5 Experimental Results

We implement our proposed hotspot detector on a platform with the Xeon Silver 4114 CPU processor and NVIDIA TITAN Xp Graphic card. We evaluate the performance of our framework on the ICCAD 2016 Benchmarks[124]. According to the results detected with the industrial $7nm$ metal layer EUV lithography simulation technique, the hotspot is precisely located. Since the first benchmark design contains

limited defects checked by lithography simulation, we only conduct the experiments on the rest three ones. For each layout, we split it into two parts with equal area size, among which one part is used for training, and the other part is used for testing. Due to the extremely large size of the whole layout, small clips are cropped from the layout and are then fed into the hotspot detector. More details are illustrated in TABLE 5.2.

Noted that `case2`, `case3` and `case4` are the names of the three benchmarks. Column "Train #HS" and "Test #HS" indicate the total number of hotspots in the training and test set, while column "Train #Clips" and "Test #Clips" refer to the total number of clips in the training and test set. "Training Set Size" and "Test Set Size" denote the resolution of the training and test set for each benchmark respectively. Clips in the training set are obtained by randomly cropping the layouts of the training part for 1000 times to get sufficient data for training. Different from the training set, we uniformly crop the layout in the test set. The size of each clip is 256×256 (corresponding to $2.56\mu m \times 2.56\mu m$), on which hotspots may appear or not.

Table 5.3 shows the results of our proposed framework and several other state-of-the-art hotspot detectors. "TCAD'19" lists the result of a deep learning-based hotspot detector proposed in [71] that adopts a high-dimensional feature extraction method and biased learning algorithm which can reduce the size of training instances. "DAC'19" shows the result of a faster region-based hotspot detector in [59] which first proposes a detector capable of detecting multiple hotspots in a large area for each inference. The comparison results illustrate that our model has satisfactory detection accuracy on each case. Especially, the average accuracy of our framework achieves 97.31 compared to 95.88 and 89.66 for DAC'19 [59] and TCAD'19 [71], respectively. Besides, the efficiency superiority of our proposed hotspot detector can

also be noticed that it achieves 67.84× speedup compared to TCAD'19 [71], which can only detect whether the center of an input clip has a hotspot. Also, compared to DAC'19 [59], the inference speed of our model is faster because of the simpler architecture of the whole framework. Moreover, the advantage of our framework can also be noticed that it suppresses the false alarm effectively, which decreases 87.6% and 59.5% of the FA reported by TCAD'19 and DAC'19.

To investigate the behavior of our designed components, we carry out ablation studies to examine how different configurations affect performance as shown in Figure 5.6. The histogram shows that with the FPN, the detection accuracy improves significantly, which reveals the importance of merging feature maps from different layers. By detecting on the feature maps from different layers, multi-level features from the input clip could be utilized for detection, which contribute to the robustness of the framework. Besides, with the center head and corner head, we obtain 2.21% improvement on accuracy and a reduction on average False Alarm, indicating that the knowledge learned in the corner head and center head can be positively leveraged for the regression and classification for bounding boxes. In addition, with feature aggregation module, we further achieve 3.1% improvement on accuracy, demonstrating that by adopting the self-attention module, the detector learns more informative representations and further achieves better detection performance.

5.6 Summary

In this chapter, we proposed an end-to-end one-stage hotspot detection framework. We take advantage of the corner and center representation to improve both classification and localization accuracy. Our feature aggregation module provides a new way to aggregate different features and further generate the enhanced features.

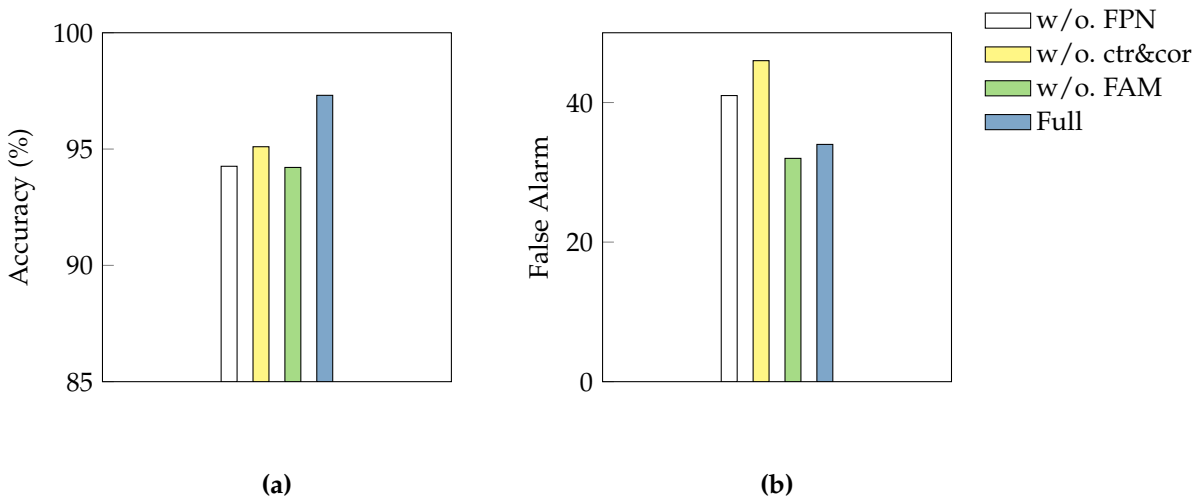


Figure 5.6: *Comparison among different configurations on (a) average accuracy and (b) average false alarm.*

We further exploit a sampling strategy for FAM to reduce the computation cost effectively. The experimental results demonstrate the superiority of our framework over current deep learning-based detectors on both accuracy and efficiency. With the development of manufacturing techniques for semiconductors, layouts are becoming more and more complex. We hope the framework proposed in this work can provide a more powerful solution to advanced design for manufacturability research.

Chapter 6

Bridging Hotspot Detection and Mask Optimization via Domain-Crossing Masked Layout Modeling

6.1 Introduction

In chip manufacturing, due to the proximity effect [125] during the lithography process, mask optimization techniques, including optical proximity correction (OPC), are first adopted to obtain the desired mask patterns, which facilitates the reduction of printing errors. Inverse lithography technology (ILT) [126] is a mathematically rigorous approach that optimizes the shapes on the mask to achieve the desired wafer patterns. It has been explored and developed as the new generation of mask optimization techniques, which is expected to solve the challenges of advanced technology nodes such as extreme ultraviolet (EUV) [127]. However, traditional ILT algorithms [17, 21, 128] typically adopt iterative methods such as gradient descent to optimize the misfit between the printed image on the wafer and the

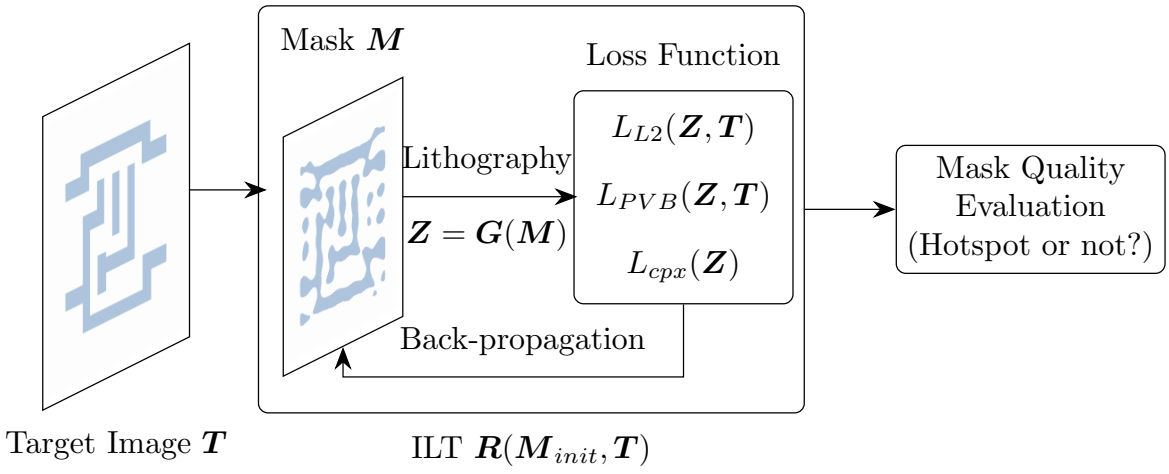


Figure 6.1: *Traditional mask optimization and hotspot detection process, where detecting hotspot regions has to rely on the result of previous mask optimization.*

target pattern, and the whole process requires many iterations for convergence, suffering from runtime overhead. To alleviate this problem, researchers propose deep learning-based algorithms to boost the efficiency of ILT and these works try to learn an ILT solver using stacked convolutional layers. By fully utilizing the massive computation resources of GPU, they can output the optimized mask patterns within a short runtime.

Although considerable progress has been made in mask optimization, there may still exist hotspots that can potentially lead to open or short failures. Current deep learning-based hotspot detection models have demonstrated their strong performance by automatically extracting key features from layout patterns and shown strong generalization ability. However, there are still some issues with learning-based methods. (1) Current deep learning-based hotspot detection models are usually trained and tested on small labeled datasets, such as ICCAD 2012 benchmark [129]. These small datasets lead to a data-hungry problem in academic research of EDA and may easily cause over-fitting of learning models. Therefore,

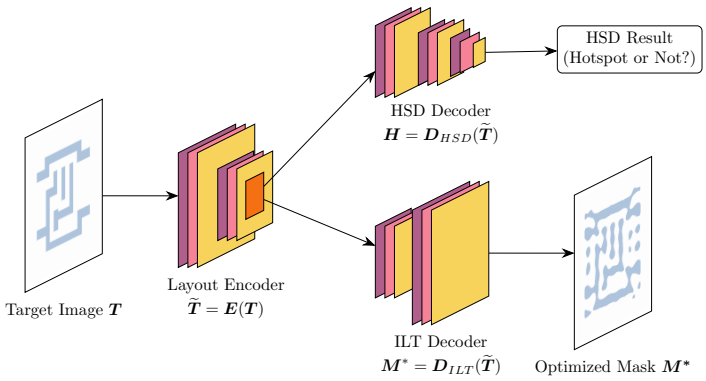


Figure 6.2: *Our proposed unified framework. A pre-trained layout-understanding encoder can boost performance on both OPC and HSD.*

learning models trained by these datasets usually have poor performance when transferred to real industrial scenarios. (2) As mentioned above, hotspot detection is closely related to mask optimization since the quality of mask optimization will determine whether any hotspot exists. However, these methods simply regard hotspot detection as an image classification problem and do not effectively utilize any prior knowledge of mask optimization and lithography process.

To improve the generalization ability, we aim to design a large pre-trained model, which can leverage more training data. Such a model can be easily transferred to multiple downstream EDA tasks. Considering both hotspot detection and mask optimization call for the understanding and knowledge of the layout geometry and the lithography process, we argue that hotspot detection and mask optimization should be integrated into a unified deep learning model, as shown in Figure 6.2. Based on these motivations, a strong pre-trained layout-understanding model (LUM) is proposed, where we design a customized training scheme called domain-crossing masked layout modeling. The proposed framework is inspired by the large pre-trained model in deep learning, such as mask auto-encoder (MAE) [130] and generative pre-trained transformer (GPT) [131], which achieves incredible performance

on many downstream tasks. In LUM, we first generate a large amount of layout data and randomly mask a portion of the layout tile. Then, a training scheme called domain-crossing mask layout modeling is proposed to guide the model training, which is illustrated in Figure 6.3. It can be seen that LUM is responsible for restoring both target images and ILT results from the given masked images. In this way, we can not only leverage massive masked layout data to pre-train the model sufficiently but also embed the awareness of the lithography process and mask optimization into LUM. After pre-training, we can easily fine-tune the pre-trained LUM with fewer data while achieving satisfactory performance on various layout understanding tasks, including both OPC and HSD. The main contributions of this chapter are listed as follows:

- We unify mask optimization and hotspot detection through the proposed pre-trained layout-understanding model.
- We create the SynLayout dataset, a large layout dataset, using layout generation techniques, which solves the data-hungry issue.
- Our proposed learning scheme, masked layout modeling, helps LUM better capture the geometric information of layout patterns, which contributes to the ability of mask optimization and hotspot detection.
- Experimental results show that our pre-trained LUM model achieves remarkable performance when fine-tuned on both OPC and HSD tasks.

The rest of this chapter is organized as follows. Section 6.2 introduces preliminaries about OPC, HSD, and mask modeling. Section 6.3 elaborates the layout understanding model with customized domain-crossing mask modeling and target reconstruction mechanism. Section 6.4 details our generated large layout dataset and experimental results, followed by the conclusion in Section 6.5.

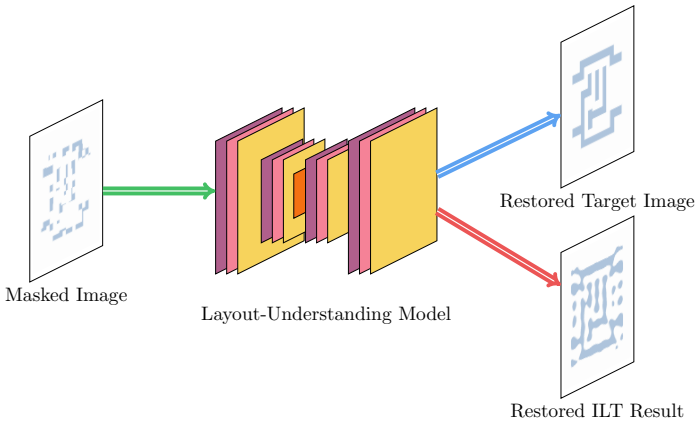


Figure 6.3: *Illustration of domain-crossing masked layout modeling. It restores the desired patterns from a masked layout tile.*

6.2 Preliminaries

In this section, we will introduce some preliminary knowledge related to this work.

6.2.1 Masked Modeling

Masked modeling is such a learning task: masking a portion of input contents and attempting to restore the contents hidden by the mask [132]. Masked language modeling, including BERT [133] and GPT [131], is the first successful application of masked modeling in natural language processing. Typically, it is a fill-in-the-blank self-supervised learning task, where a model learns representations by predicting what a masked word should be according to the context words surrounding the token. Recently, masked image modeling [130, 134] follows a similar way to learn representations by predicting the missing parts at the pixel or patch level. Once the model is trained and evaluated, it can be deployed in production systems for various applications, such as object detection, instance segmentation, semantic segmentation, or medical image analysis. Mask modeling plays a crucial role in computer vision

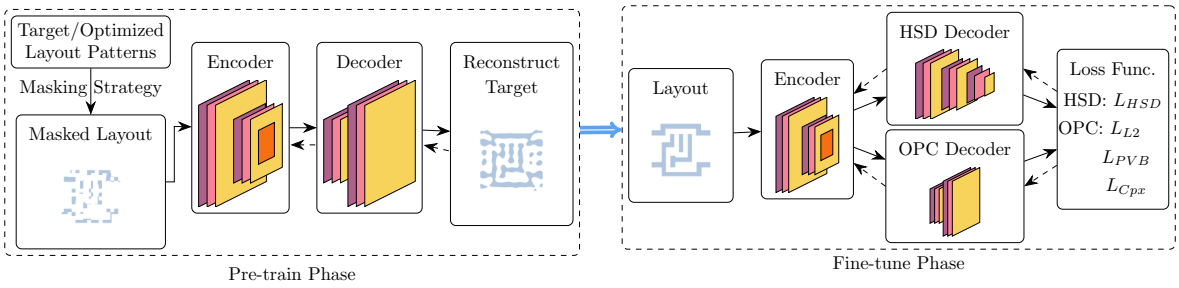


Figure 6.4: *Overview of the proposed framework, consisting of the pre-training, fine-tuning, and co-evolution phases. The solid and dashed lines indicate the feed-forward and back-propagation in training, respectively.*

tasks where precise localization and pixel-level understanding of objects or regions are required.

Nonetheless, the techniques described above have only been shown to be useful for natural language and image modeling. Masked modeling has not been fully explored in the EDA domain. It is widely known that there are many layout-related tasks that call for strong layout understanding abilities of the deep learning model. In this work, we aim to investigate the application of masked modeling for more robust layout feature extraction. We mask a portion of the layout, and our layout-understanding model is responsible for restoring the masked layout image. In addition, regarding the uniqueness of EDA layouts and tasks, we have made many customizations to the mask layout modeling mechanism, which is significantly different from the mask modeling mechanisms in computer vision and natural language processing fields, which will be detailed in Section 6.3.5.

6.3 Algorithms

In this section, the architecture of our layout understanding model (LUM) will be explained in detail. We will introduce the critical components that enable the LUM

to be equipped with strong generalization abilities that differ LUM from previous attempts on layout feature learning models. We also show how to incorporate the prior knowledge of mask optimization into LUM, which remarkably contributes to the HSD and OPC performance.

6.3.1 Overview

As shown in Figure 6.4, the proposed framework consists of two phases:

Pre-training Phase: . In this phase, we employ the proposed domain-crossing masked layout modeling method to pre-train the model. The masking strategy (Section 6.3.2) masks a specific portion of input patches, the LUM encoder (Section 6.3.3) maps the input patches to the latent space, and the LUM decoder (Section 6.3.4) aims to recover the desired layout patterns. During the pre-training stage, the inputs and outputs can be target images or optimized masks. This training scheme enables the model to acquire knowledge of the two domains. The reconstruction target (Section 6.3.5) guides the pre-training process via back-propagation.

Fine-tuning Phase: . Since our framework is targeted at OPC and HSD, we need an OPC decoder and an HSD decoder to accomplish these two tasks according to the encoding from the LUM encoder. We fine-tune the LUM decoder used in the pre-train phase to build the OPC decoder. In addition, we introduce a new HSD decoder for classification. Section 6.3.6 demonstrates how to fine-tune the model on these tasks.

6.3.2 Masking Strategy

Before training our LUM framework to restore the layout pattern, the first step is to mask a portion of the layout tile suitably. Most recent mask modeling approaches, such as MAE [130] and SimMIM [134], followed a uniformly random masking method at the patch level. Inspired by these works, the layout pattern in our work is first divided into non-overlapping patches of size 4×4, and part of the patches are randomly masked.

The masking ratio is critical for model training. A suitable ratio can ensure that the masking layout patterns effectively eliminate redundancy, resulting in a task that cannot be solved by extrapolation from visible neighboring patches. In previous vision works, a commonly adopted masking ratio is 75%. However, we find that such a high ratio is not suitable for our task since the meaning of semantic information of layout patterns is more sparse than in regular images. To determine a suitable masking ratio, a series of experiments are investigated, and we finally set the masking ratio as 50%.

6.3.3 Encoder

The LUM encoder is responsible for modeling latent feature representations of the masked patches, which are then utilized to forecast the original signals in the masked area. The learned encoder should be capable of adapting to different tasks.

Recently, Transformer [78] has become a powerful encoder in both vision and language areas. The transformer encoder consists of multiple layers, of which the most important one is the multi-head self-attention, allowing the model to capture information at different positions globally [78]. To deal with image inputs, Vision Transformer (ViT) [135] splits an image into fixed-size patches, each of

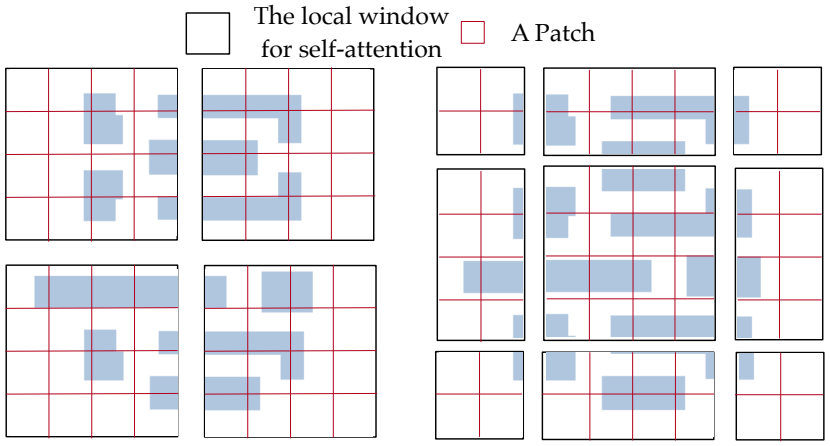


Figure 6.5: *The shifted window approach for computing self-attention in our proposed encoder architecture. In layer W-MSA, we adopt a regular window partitioning scheme as shown in the left part, and self-attention is computed within each local window. In layer SW-MSA, the window is shifted as shown in the right part.*

which is regarded as a token in sequential data. Given an input layout of size $H \times W \times C$ and the patch size P, the representation is first transformed into patches $\{\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_{\frac{HW}{P^2}}\}$.

The Transformer encoder first packs the patches as a matrix, represented as $\vec{X} \in \mathbb{R}^{\frac{HW}{P^2} \times CP^2}$. Next, a fully-connected layer maps the patches to the input embeddings $\vec{X}_E \in \mathbb{R}^{\frac{HW}{P^2} \times N_H}$. To differentiate the positions of patches, positional encodings $\vec{E}_P \in \mathbb{R}^{\frac{HW}{P^2} \times N_H}$ are then added to the input embeddings.

$$\vec{X}_P = \vec{X}_E + \vec{E}_P \tag{6.1}$$

However, the vanilla Transformer [78] is extremely computationally expensive. Inspired by Swin Transformer [136], our encoder module introduces three extra mechanisms to further improve the efficiency while keeping the advantage of the Transformer [78]. The first one is the local attention mechanism. Instead of applying attention to all patches, we group the patches into $W \times W$ windows, and a multi-head self-attention layer is employed within each window. Such a

mechanism is implemented as a window multi-head self-attention (W-MSA) layer. Local attention improves computation efficiency but lacks global perception. The second mechanism solves this problem by introducing shifted windows to partition the original layout pattern. Specifically, the local attention mechanism uses a regular window partitioning strategy that starts from the top-left, while the shifted window mechanism displaces the windows by $(\lfloor \frac{H}{2} \rfloor, \lfloor \frac{W}{2} \rfloor)$. Figure 6.5 illustrates the cross-connection ability of the shifted windows mechanism. This mechanism is called a shifted window multi-head self-attention (SW-MSA) layer. Furthermore, the patch merging mechanism downscales the features by concatenating neighbouring patches, which can enable the aggregation of multiscale information. Our encoder contains multiple stages, and Figure 6.6 presents a single stage, which consists of a patch merging layer and N_i Swin Transformer blocks. Each Swin Transformer block includes W-MSA, MLP, SW-MSA, and MLP layers, where MLP means multi-layer perceptrons.

6.3.4 Decoder

The LUM decoder is designed based on the feature pyramid network (FPN) [137], which is a widely used and effective technique in the field of computer vision. FPN serves as a versatile feature extractor that capitalizes on the inherent and pyramidal hierarchy of features. This allows for the extraction of multi-level feature representations.

In our framework, the LUM decoder takes as input the output features obtained from the four stages of the LUM encoder depicted in Figure 6.7. These features serve as the foundation for the subsequent decoding process. To generate feature maps with varying levels of detail, the LUM decoder employs a top-down pathway. This pathway utilizes convolutional layers, residual connections, and a pyramid

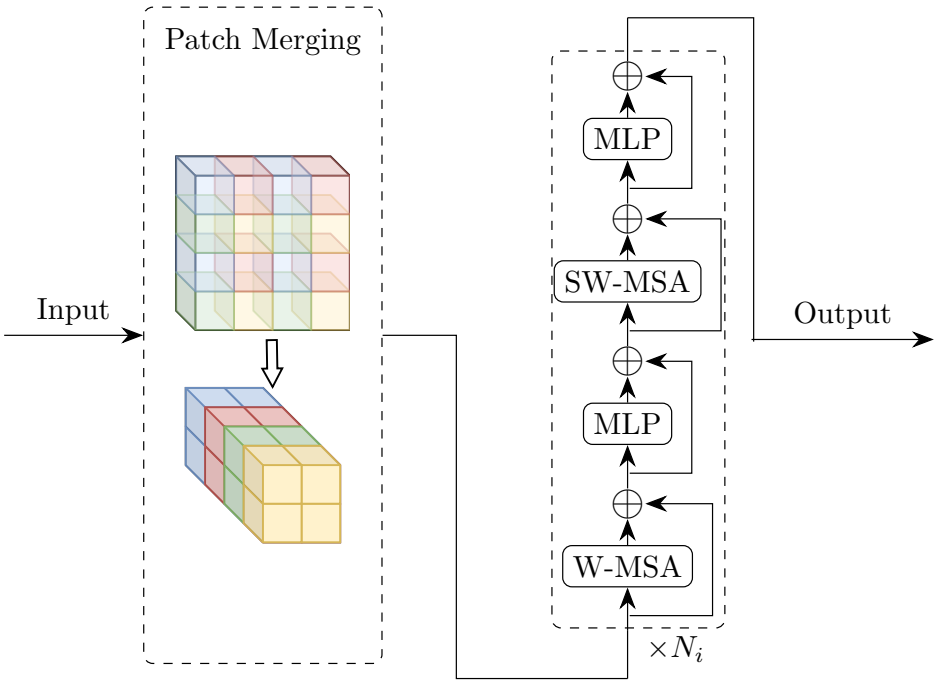


Figure 6.6: *Illustration of a stage in the Swin Transformer architecture, which consists of a patch merging layer and N_i Swin Transformer blocks.*

pooling module [138] to process the input feature maps. The result is the creation of multi-scale feature maps, each containing features at different levels of abstraction.

In addition, lateral connections are introduced to integrate information from different levels of the feature hierarchy. These connections combine feature maps from different levels and facilitate the creation of a single consolidated feature map. This final feature map is then fed through readout convolutional layers to produce the ultimate output, which represents the generated layout.

6.3.5 Target Reconstruction

Our model is able to receive or output target images or optimized masks. For example, receiving the target image and generating the optimized mask is equivalent to the OPC. Receiving the masked target image and generating the restored

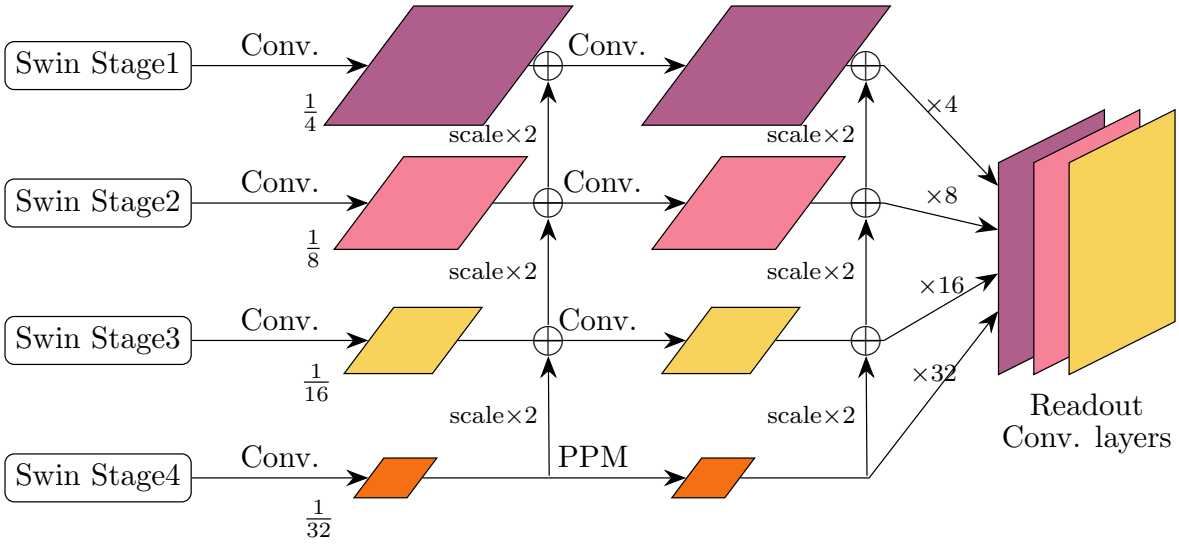


Figure 6.7: *LUM decoder. The inputs to the LUM decoder are the output features from the four stages of the LUM encoder. With a top-down pathway, multi-scale feature maps can be generated, each containing different level features.*

target image is the layout reconstruction process. Combining different input and reconstruction targets enables our LUM to acquire knowledge of different domains.

To distinguish between target images and optimized tasks, we design an additional type of embedding for the LUM encoder. Specifically, given the input embeddings $\vec{X}_E$, we apply the type embedding by:

$$\vec{X}_P = \begin{cases} \vec{X}_E + \vec{E}_T, & \text{if the input is a target image,} \\ \vec{X}_E + \vec{E}_M, & \text{if the input is an optimized mask.} \end{cases} \tag{6.2}$$

Similarly, we also introduce the type embedding to the LUM decoder. The embedding matrix $\widetilde{\vec{E}}_T$ or $\widetilde{\vec{E}}_M$ is added to the input of the MLM decoder. We set the type embedding matrices $\vec{E}_T$, $\vec{E}_M$, $\widetilde{\vec{E}}_T$, and $\widetilde{\vec{E}}_M$ trainable. This mechanism enables the encoder and decoder to differentiate between the two domains.

During the pre-training stage, when given a layout tile $\vec{X}$, we first obtain its corresponding target image $\vec{X}_T$ and optimized mask $\vec{X}_M$. Next, we mask the images

and get $\overline{\vec{X}}_T$, $\overline{\vec{X}}_M$. Denoting the mask layout modeling with $\vec{f} : \mathbb{R}^{H\times W} \to \mathbb{R}^{H\times W}$, the reconstruction target can be formulated as:

$$\begin{aligned} L_{cross}(\vec{X}) = \|\vec{f}_{TT}(\overline{\vec{X}}_T) - \vec{X}_T\|_2^2 + \|\vec{f}_{TM}(\overline{\vec{X}}_T) - \vec{X}_M\|_2^2 + \\ \|\vec{f}_{MT}(\overline{\vec{X}}_M) - \vec{X}_T\|_2^2 + \|\vec{f}_{MM}(\overline{\vec{X}}_M) - \vec{X}_M\|_2^2. \end{aligned} \tag{6.3}$$

In Equation (6.3), the two subscripts of $\vec{f}$ indicate the input and output domains. For example, $\vec{f}_{TM}$ uses $\vec{E}_T$ as the encoder type embedding and $\widetilde{\vec{E}}_M$ as the decoder type embedding.

6.3.6 Fine-tuning

In the fine-tuning phase, we adapt the pre-trained model to HSD and OPC tasks. For OPC, we simply fine-tune the pre-trained mask layout modeling decoder to predict the optimized masks. Given the target image $\vec{X}_T$ and the optimized mask $\vec{X}_M$, the loss function is defined as:

$$L_{OPC}(\vec{X}_T, \vec{X}_M) = L_M(\vec{X}_T, \vec{X}_M) + L_{L2}(\vec{X}_T) + L_{PVB}(\vec{X}_T). \tag{6.4}$$

The loss function (6.4) involves the following components:

- The similarity between the predicted mask and the optimized mask is minimized by:

$$L_M(\vec{X}_T, \vec{X}_M) = \|\vec{f}_{TM}(\overline{\vec{X}}_T) - \vec{X}_M\|_2^2. \tag{6.5}$$

- The L_2 loss minimizes the distance between the printed image and the target image, which is formulated as:

$$L_{L2}(\vec{X}_T) = \|\vec{f}_Z(\vec{f}_{TM}(\overline{\vec{X}}_T)) - \vec{X}_T\|_2^2. \tag{6.6}$$

  The printed image $\vec{f}_Z(\vec{f}_{TM}(\overline{\vec{X}}_T))$ is computed by the lithography simulation

model.

- $L_{PVB}(\vec{X}_T)$ improves the robustness of the OPC results by minimizing the distance between the printed images at the maximum and minimum process corners. Specifically, the maximum and minimum process corners are modeled by lithography kernels $\vec{H}_{max}$ and $\vec{H}_{min}$, respectively. Given the printed images $\vec{f}_{Z_{max}}(\vec{f}_{TM}(\overline{\vec{X}}_T))$ and $\vec{f}_{Z_{min}}(\vec{f}_{TM}(\overline{\vec{X}}_T))$, the PVB loss is defined as:

$$L_{PVB}(\vec{X}_T) = \|\vec{f}_{Z_{max}}(\vec{f}_{TM}(\overline{\vec{X}}_T)) - \vec{f}_{Z_{min}}(\vec{f}_{TM}(\overline{\vec{X}}_T))\|_2^2. \tag{6.7}$$

Since the objective of HSD is very different from the OPC task, a customized HSD decoder is specifically designed to output whether there exist hotspot regions in the layout patterns. The HSD decoder is composed of three consecutive convolution layers, which receive the output of the previous encoder and predict whether there exists a hotspot in each patch as divided in Section 6.3.2. Therefore, the output of the HSD decoder is an $N \times N$ feature map, where each pixel value denotes the probability of the hotspot region. To guide the learning of the HSD decoder, we adopt a loss function based on the cross-entropy loss, which is defined as:

$$L_{HSD}(\vec{X}_T) = \sum_{x=1}^{N}\sum_{y=1}^{N} -p_{xy}\log\hat{p}_{xy} - (1-p_{xy})\log(1-\hat{p}_{xy}), \tag{6.8}$$

where p_{xy} and $\hat{p}_{xy}$ denote the ground truth and prediction probability of the hotspot region respectively.

6.4 Experiments

We implement our entire framework LUM with the widely used deep learning library, Pytorch [104]. The model is tested on a Linux system with a 2.3GHz Intel

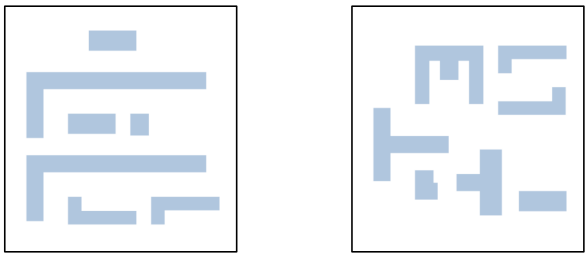


Figure 6.8: *Examples of Synthetic Layout Patterns.*

Xeon CPU and a single NVIDIA GeForce RTX 3090 GPU. To verify the efficiency and robustness of our approach, we synthesized a large layout dataset, called SynLayout Dataset, to test the performance of LUM.

6.4.1 SynLayout Dataset

In previous works, the ICCAD-2012 benchmark [129] and ICCAD-2013 [102] are two commonly used layout datasets for hotspot detection and mask optimization. However, these benchmarks are only used for academic research, and they contain very limited layouts. To be specific, ICCAD-2013 [102] only contains ten $2\mu m \times 2\mu m$ metal layer clips, which are too small to fit AI solutions. To create a larger dataset, GAN-OPC [19] releases around 4k synthetic tiles of metal-layer. However, its scale still cannot meet the training requirements of deep learning models currently applied in layout-related tasks. Previous learning models trained by these small datasets usually have unsatisfactory performance when applied in real industrial scenarios due to weak generalization ability.

In order to further improve the generalization ability of our model, we are supposed to generate and leverage more layout pattern data to train our LUM effectively. Therefore, we create a new layout dataset called SynLayout, where we synthesize 16000 tiles following the layout generation method proposed in [139]. Each tile is

Table 6.1: *Benchmark information of our SynLayout dataset.*

Benchmark	SynLayout
Train #Layout	11200
Test #Layout	4800
Train #HS	43269
Test #HS	18120

Table 6.2: *Comparison with state-of-the-art HSD methods on SynLayout dataset.*

	Accuracy (%)	False Alarm	Runtime (s)
TCAD'19 [71]	85.63	6384	0.40
DAC'19 [59]	89.76	4629	0.28
ICCAD'21 [141]	91.94	3241	**0.12**
Ours	**94.21**	**2860**	0.22

randomly generated according to the design rules of ICCAD-13 benchmark. The rules include the requirements on shape widths, distances, and areas, determined by the technology node. We split 70% of the data for training and 30% of the data for testing.

In addition, we also need to obtain the corresponding optimized masks and hotspot regions of all generated layout patterns, which act as supervised signals. Traditional optimization-based ILT methods are adopted to obtain the accurate optimized masks of our generated layouts. L2 loss (Equation (6.6)), PVB loss (Equation (6.7)), and curvature loss [140] are employed in the ILT method to ensure the mask quality while minimizing the mask complexity. After obtaining all optimized masks, we sample a series of points and calculate their EPE values, and the locations that have EPE violations are determined as hotspot regions. Figure 6.8 shows synthetic layout pattern examples which follow legal design rules. Details of our dataset are listed in Section 6.4.1, where columns "Train #HS" and "Test #HS" indicate the total number of hotspots in the training and test set.

Table 6.3: *Mask printability comparison with sate-of-the-art methods on ICCAD 2013 benchmark.*

Benchmarks	MOSAIC [126]			DevelSet [140]			CTM-SRAF [142]			Ours-Fast			Ours-Exact		
	EPE	L_2 (nm^2)	PVB (nm^2)	EPE	L_2 (nm^2)	PVB (nm^2)	EPE	L_2 (nm^2)	PVB (nm^2)	EPE	L_2 (nm^2)	PVB (nm^2)	EPE	L_2 (nm^2)	PVB (nm^2)
case1	6	49893	65534	10	49142	59607	3	46405	57584	5	46655	47519	3	39561	47788
case2	10	50369	48230	1	34489	52010	1	33481	45756	0	33893	37084	0	31261	38279
case3	59	81007	108608	64	93498	76558	28	77734	92660	40	80918	76726	15	63798	76121
case4	1	20044	28285	2	18682	29047	0	13183	26061	1	12014	22039	0	8939	23679
case5	6	44656	58835	1	44256	58085	1	41569	54553	0	36695	55481	0	30208	53504
case6	1	57375	48739	2	41730	53410	1	38608	48134	0	37338	47979	0	30284	47809
case7	0	37221	47120	0	35329	46071	0	32443	43697	0	30640	43268	0	28579	43012
case8	2	19782	22846	0	15460	24836	1	15178	20657	0	12751	20535	0	10813	20192
case9	6	55399	66331	0	50834	64950	0	49073	60754	0	42860	58123	0	34738	60962
case10	0	24381	18097	0	10140	21619	0	8231	17426	0	10323	16544	0	7714	16234
Average	9.1	44012	50899	8	38402	48672	3.5	34765	46753	4.6	33292	**42179**	**1.8**	**27349**	42577

Table 6.4: *Runtime comparison with state-of-the-art methods on ICCAD 2013 benchmark.*

Benchmarks	MOSAIC [126]	DevelSet [140]	CTM-SRAF [142]	Ours-Fast	Ours-Exact
case1	318	1.50	121	0.01	6.6
case2	256	1.40	93	0.01	6.6
case3	321	1.29	179	0.01	6.6
case4	322	1.65	128	0.01	6.6
case5	315	0.91	73	0.01	6.6
case6	314	0.84	72	0.01	6.6
case7	239	0.76	78	0.01	6.6
case8	258	1.14	66	0.01	6.6
case9	322	1.21	74	0.01	6.6
case10	231	0.42	57	0.01	6.6
Average	289	1.1	94.1	**0.01**	6.6

6.4.2 Results Comparison

Table 6.2 shows the comparison hotspot detection results of our proposed framework and several other state-of-the-art hotspot detectors. The comparison results

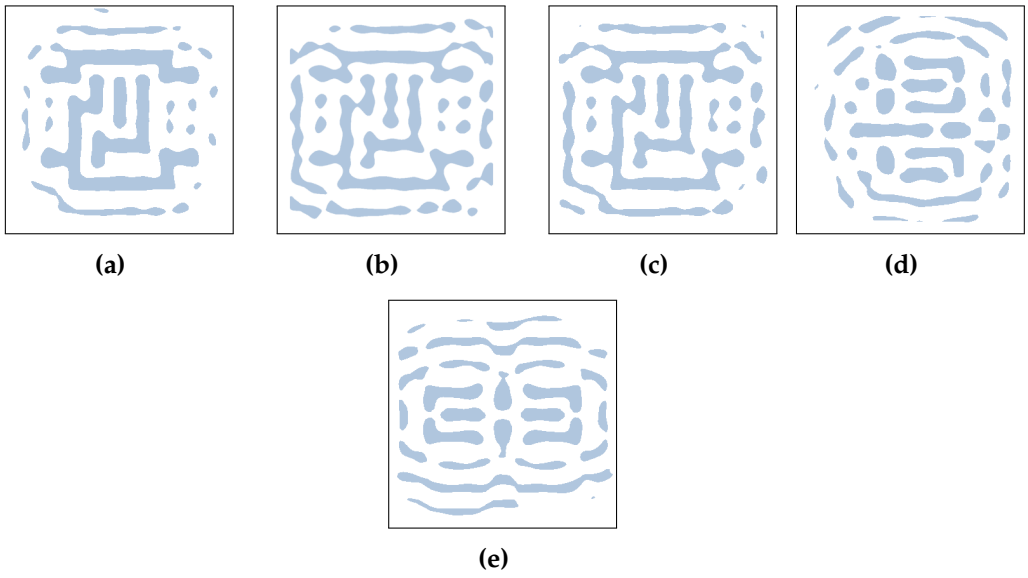


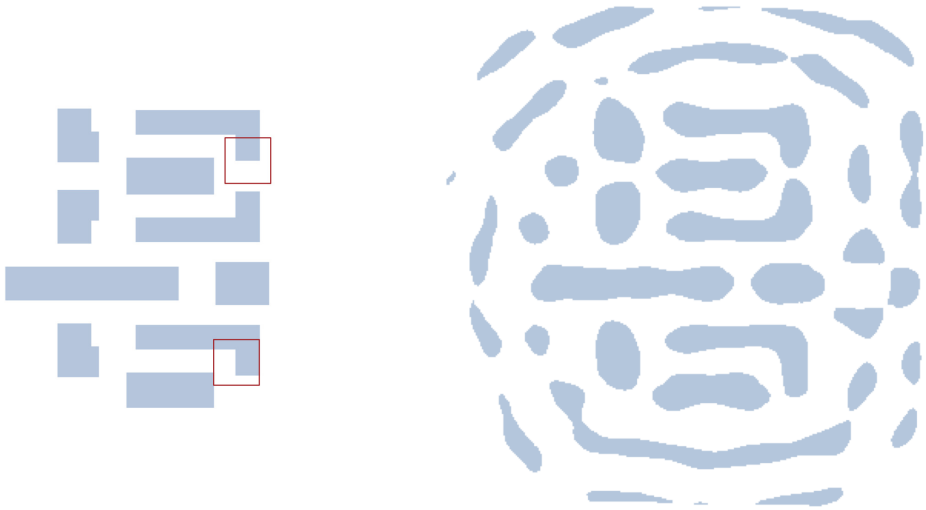
Figure 6.9: *Examples of OPC results. (a) shows a mask output by the pre-trained LUM without further training. (b) is a well-optimized mask from the "Ours-Exact" setting. (c) (d) (e) present three examples from the "Ours-Fast" setting.*

illustrate that our model LUM has satisfactory performance. Specifically, the average accuracy of our framework is 94.21% compared to 91.94%, 89.76%, and 85.66% for ICCAD'21 [141], DAC'19 [59], and TCAD'19 [71], respectively. Besides, the advantage of our framework can also be noticed that it suppresses the false alarm effectively, which decreases 55.2%, 30.0%, and 11.8% of the false alarm reported by TCAD'19 [71], DAC'19 [59] and ICCAD'21 [141]. As for runtime, it averagely takes 0.22s for LUM to detect a hotspot on a single layout, which is a little bit slower than ICCAD'21 [141] (0.12s) and faster than the other two works. This is because our model is designed for multiple tasks, including both hotspot detection and mask optimization, and the architecture of LUM is more complex than ICCAD'21 [141]. In contrast, the model in ICCAD'21 [141] is only used for hotspot detection and has a simple structure. However, we think the runtime is still comparable, and it is worth achieving much better performance at the cost of extra limited time.

6.4.3 What does LUM learn?

Our motivation for designing LUM is to incorporate the prior knowledge of mask optimization into the model, which is beneficial for improving the accuracy of hotspot detection. To prove that LUM has also learned the process of OPC, we concatenate the encoder and the OPC decoder and ask LUM to receive the target image and directly output the optimized masks without any fine-tuning process. Figure 6.9a presents an example of the pre-trained results. Comparing Figure 6.9a to Figure 6.9b, the generated masks without OPC-specific training are close to the well-optimized masks. This observation indicates that the pre-trained LUM can be equipped with OPC-based knowledge, which contributes to the outstanding HSD results.

To provide quantitative analysis, we further conduct OPC-specific training. The methods are tested on ICCAD 2013 benchmark [102]. Table 6.3 and Table 6.4 compare the mask printability and runtime performance of various OPC methods. "MOSAIC" [126] is a pixel-based ILT method. "DevelSet" [140] is a level set-based ILT method that incorporates DNN models to reduce the optimization time. "CTM-SRAF" [142] combines the advantages of pixel-based and level set-based methods to achieve better performance while maintaining satisfactory mask complexity. "Ours-Fast" shows the performance of the masks generated by our OPC decoder. Figure 6.9c, Figure 6.9d, and Figure 6.9e present the masks generated by "Ours-Fast". "Ours-Exact" is obtained by fine-tuning the masks with our optimization-based ILT method. Figure 6.9b shows the fine-tuned version of Figure 6.9c. It can be seen that "Ours-Fast" achieves better L2 and PVB than the baselines with a significantly lower runtime. With acceptable runtime overhead, "Ours-Exact" not only improves the L2 and PVB but also gets remarkable EPE results. The superior performance indicates



(a) (b)

Figure 6.10: *HSD and OPC results of one layout. (a) shows the hotspot detection result (b) is the corresponding mask optimization result.*

that our LUM-based OPC can achieve better manufacturability. In summary, "Ours-Fast" achieves instant mask optimization with nice results, while "Ours-Exact" attains outstanding performance with acceptable runtime. These results not only validate the effectiveness of the proposed layout-understanding model, but also highlight the importance of the co-evolution of HSD and OPC.

Finally, we also visualize the results in Figure 6.10a and Figure 6.10b obtained by applying both the OPC decoder and the HSD decoder to the same layout, aiming to demonstrate the detected hotspot regions correspond to the potential open-circuit-prone or short-circuit-prone regions in the optimized mask. It can be observed that as the light intensity increases, there is a significant possibility of short-circuit occurrence at the highlighted regions on the optimized mask. This further demonstrates the accuracy of our LUM framework for both OPC and HSD.

6.5 Summary

In this work, we present a layout understanding model (LUM), a deep learning-based model that achieves remarkable performance on both mask optimization and hotspot detection tasks. Our model structure is inspired by the mask modeling method, which masks a portion of input signals and asks the model to restore the content. Such a model can leverage more training data to improve the generalization ability. We propose different decoders for HSD and OPC tasks while maintaining the same encoder component. During the training stage, we design multiple reconstruction tasks to enable the model to effectively learn the geometric information of layout patterns as well as the process of mask optimization. The experimental results demonstrate that our model has shown remarkable performance on OPC tasks, and the learned mask optimization knowledge is also beneficial for improving the HSD performance.

Chapter 7

Conclusion

In the thesis, several machine learning-driven methodologies are introduced to solve several typical physical verification tasks in design for manufacturability. Our major contributions include:

- In Chapter 3, we propose an efficient framework for automating Design Rule Checking (DRC) script generation. By leveraging deep learning-based key information extraction, the framework significantly reduces the turnaround time and improves the accuracy of rule extraction. The experimental results demonstrate its effectiveness in generating scripts and its potential to streamline the DRC process. Overall, DRC-SG represents a valuable contribution to electronic design automation (EDA) with its automation capabilities and adaptability to different design rule checkers.

- In Chapter 4, we introduce L2O-ILT, a deep learning-based framework for inverse lithography technique (ILT) optimization. L2O-ILT generates a high-quality initial mask, enabling fast refinement and improving mask printability while reducing runtime. The framework combines the strengths of conventional ILT and "generative ILT" approaches, offering interpretability,

domain-specific knowledge, and efficient solution search. Experimental results demonstrate its effectiveness in accelerating ILT and enhancing mask quality, opening new possibilities for efficient VLSI design.

- In Chapter 5, we present an innovative approach for hotspot detection using multi-task learning and Transformer Encoder. The proposed single-stage detector eliminates the need for a region proposal stage, improving efficiency. The center and corner head modules enhance accuracy, while the feature aggregation module captures feature relationships. Experimental results demonstrate superior accuracy and speed compared to previous models. This work contributes valuable advancements to hotspot detection in semiconductor manufacturing, offering an efficient and accurate solution for practical implementation.

- In Chapter 5, we design a unified deep-learning model, the layout understanding model (LUM), that integrates hotspot detection and mask optimization. By pre-training LUM on a large layout dataset using domain-crossing masked layout modeling, the model achieves remarkable performance on both tasks. The proposed approach improves efficiency, reduces runtime overhead, and addresses data scarcity. This research has significant implications for enhancing chip manufacturing processes and advancing technology nodes.

Despite the exploration of new methodologies in this thesis, there remain significant challenges in design automation and system integration. As the complexity of design, manufacturing, and integration continues to grow, it is anticipated that additional methodologies will emerge to drive further technological innovation. Specifically, the exploration of the following research problems and directions holds great value.

- For the generation of DRC scripts, it is necessary to explore more efficient flows. For instance, we can investigate the possibility of directly generating scripts without the need for extracting key information and then translating them in a two-stage flow. Additionally, we need to collect a wider range of rule categories for advanced nodes. Currently, the number and types of design rules available in the academic community are significantly fewer compared to those in the industry. In the future, we need to consider how to improve the accuracy of script generation for more complex rules in the industry.

- With the advancement of chip technology, the size of layouts is increasing. Existing optical proximity correction (OPC) techniques are typically designed for small-scale layouts. However, there is a need to explore how to migrate these existing techniques to large-scale layouts. This involves the partitioning and stitching of layouts to ensure seamless integration after performing OPC on the divided sections. Therefore, it is necessary to explore more robust partitioning and stitching techniques to enable the efficient handling of large-scale layouts.

- Regarding the hotspot detection, one potential direction is to introduce multi-modal learning in the field of hotspot detection. With the rapid development of large language models, they have already achieved success in various applications within electronic design automation (EDA), such as Verilog generation. In the future, we can also consider leveraging the GDS representation of patterns during the hotspot detection stage. This representation includes the coordinates of all points in each polygon of the pattern. By simultaneously utilizing large language models and convolutional generative networks, we can analyze both the GDS files and images, thereby enhancing the accuracy of

hotspot detection.

References

[1] George E Bier and Andrew R Pleszkun. "An algorithm for design rule checking on a multiprocessor". In: *22nd ACM/IEEE Design Automation Conference*. IEEE. 1985, pp. 299–304.

[2] SK Nandy et al. "Geometric Design Rule Check of VLSI Layouts in Distributed Computing Environment". In: *VLSI Design* 1 (1994), pp. 155–167.

[3] F Gregoretti and Z Segall. "Analysis and evaluation of VLSI design rule checking implementation in a multiprocessor". In: *Proc. Int. Conf. Parallel Processing*. 1984, pp. 7–14.

[4] Zhuolun He et al. "OpenDRC: An Efficient Open-Source Design Rule Checking Engine with Hierarchical GPU Acceleration". In: *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE. 2023, pp. 1–6.

[5] Bentley and Wood. "An optimal worst case algorithm for reporting intersections of rectangles". In: *IEEE Transactions on Computers* 100.7 (1980), pp. 571–577.

[6] Glenn G Lai, Donald S Fussell, and DF Wong. "Hinted quad trees for VLSI geometry DRC based on efficient searching for neighbors". In: *IEEE transactions on computer-aided design of integrated circuits and systems* 15.3 (1996), pp. 317–324.

[7] David Marple, Michiel Smulders, and Henk Hegen. "Tailor: A layout system based on trapezoidal corner stitching". In: *IEEE transactions on computer-aided design of integrated circuits and systems* 9.1 (1990), pp. 66–90.

[8] Antonin Guttman. "R-trees: A dynamic index structure for spatial searching". In: *Proceedings of the 1984 ACM SIGMOD international conference on Management of data*. 1984, pp. 47–57.

[9] Luis Francisco et al. "Design rule checking with a CNN based feature extractor". In: *Proceedings of the 2020 ACM/IEEE Workshop on Machine Learning for CAD*. 2020, pp. 9–14.

[10] Wei Zeng, Azadeh Davoodi, and Rasit Onur Topaloglu. "Explainable DRC hotspot prediction with random forest and SHAP tree explainer". In: *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2020, pp. 1151–1156.

[11] Zhiyao Xie et al. "RouteNet: Routability prediction for mixed-size designs using convolutional neural network". In: *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE. 2018, pp. 1–8.

[12] Riadul Islam and Md Asif Shahjalal. "Late Breaking Results: Predicting DRC Violations Using Ensemble Random Forest Algorithm". In: *ACM/IEEE Design Automation Conference (DAC)*. 2019, pp. 1–2.

[13] Aysa Fakheri Tabrizi et al. "A Machine Learning Framework to Identify Detailed Routing Short Violations from a Placed Netlist". In: *ACM/IEEE Design Automation Conference (DAC)*. 2018, pp. 1–6. DOI: 10.1109/DAC.2018.8465835.

[14] Jian Kuang, Wing-Kai Chow, and Evangeline F. Y. Young. "A robust approach for process variation aware mask optimization". In: *IEEE/ACM Proceedings Design, Automation and Test in Eurpoe (DATE)*. 2015, pp. 1591–1594.

[15] Yu-Hsuan Su et al. "Fast lithographic mask optimization considering process variation". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 35.8 (2016), pp. 1345–1357.

[16] Tetsuaki Matsunawa, Bei Yu, and David Z. Pan. "Optical proximity correction with hierarchical Bayes model". In: *Journal of Micro/Nanolithography, MEMS, and MOEMS (JM3)* 15.2 (2016), p. 021009.

[17] Jhih-Rong Gao et al. "MOSAIC: Mask Optimizing Solution With Process Window Aware Inverse Correction". In: *ACM/IEEE Design Automation Conference (DAC)*. San Jose, California, 2014, 52:1–52:6.

[18] Thomas Cecil et al. "Establishing fast, practical, full-chip ILT flows using machine learning". In: *Proceedings of SPIE*. 2020.

[19] Haoyu Yang et al. "GAN-OPC: Mask Optimization with Lithography-guided Generative Adversarial Nets". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*. 2020, pp. 2822–2834.

[20] Bentian Jiang et al. "Neural-ILT 2.0: Migrating ILT to Domain-Specific and Multitask-Enabled Neural Network". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* (2022).

[21] Ziyang Yu et al. "A GPU-enabled Level Set Method for Mask Optimization". In: *IEEE/ACM Proceedings Design, Automation and Test in Eurpoe (DATE)*. 2021.

[22] Guojin Chen et al. "DevelSet: Deep neural level set for instant mask optimization". In: *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 2021.

[23] Yuzhe Ma et al. “A unified framework for simultaneous layout decomposition and mask optimization”. In: *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 2017, pp. 81–88. DOI: 10.1109/ICCAD.2017.8203763.

[24] Yijiang Shen, Ngai Wong, and Edmund Y. Lam. “Level-set-based inverse lithography for photomask synthesis”. In: *Opt. Express* 17.26 (2009), pp. 23690–23701. DOI: 10.1364/OE.17.023690.

[25] Yijiang Shen et al. “Robust level-set-based inverse lithography”. In: *Optics Express* 19.6 (2011), pp. 5511–5521.

[26] Kevin Hooker et al. “ILT optimization of EUV masks for sub-7nm lithography”. In: *SPIE*. Vol. 10446. 2017, pp. 9–20.

[27] Ryan Pearman et al. “How curvilinear mask patterning will enhance the EUV process window: a study using rigorous wafer+ mask dual simulation”. In: *SPIE*. Vol. 11178. 2019, pp. 59–67.

[28] Linyong Pang. “Inverse lithography technology: 30 years from concept to practical, full-chip reality”. In: *Journal of Micro/Nanopatterning, Materials, and Metrology* 20.3 (2021), pp. 030901–030901.

[29] Shuyuan Sun et al. “Efficient ILT via Multi-level Lithography Simulation”. In: *ACM/IEEE Design Automation Conference (DAC)*. 2023.

[30] Ian Goodfellow et al. “Generative adversarial networks”. In: *Communications of the ACM* 63.11 (2020), pp. 139–144.

[31] Mehdi Mirza and Simon Osindero. “Conditional generative adversarial nets”. In: *arXiv preprint arXiv:1411.1784* (2014).

[32] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation". In: *International Conference on Medical image computing and computer-assisted intervention*. Springer. 2015, pp. 234–241.

[33] Huimin Huang et al. "Unet 3+: A full-scale connected unet for medical image segmentation". In: *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2020, pp. 1055–1059.

[34] Haoyu Yang et al. "GAN-OPC: Mask optimization with lithography-guided generative adversarial nets". In: *ACM/IEEE Design Automation Conference (DAC)*. 2018.

[35] Guojin Chen et al. "DAMO: Deep agile mask optimization for full chip scale". In: *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 2020.

[36] Bentian Jiang et al. "Neural-ILT: Migrating ILT to neural networks for mask printability and complexity co-optimization". In: *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 2020.

[37] Qijing Wang et al. "A2-ILT: GPU Accelerated ILT with Spatial Attention Mechanism". In: *ACM/IEEE Design Automation Conference (DAC)*. 2022, pp. 967–972.

[38] David Z. Pan, Bei Yu, and J.-R. Gao. "Design for Manufacturing With Emerging Nanolithography". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 32.10 (2013), pp. 1453–1472.

[39] Bei Yu et al. "Design for manufacturability and reliability in extreme-scaling VLSI". In: *Science China Information Sciences* 59 (2016), pp. 1–23.

[40] Juhwan Kim and Minghui Fan. “Hotspot detection on post-OPC layout using full-chip simulation-based verification tool: a case study with aerial image simulation”. In: *23rd Annual BACUS Symposium on Photomask Technology*. Vol. 5256. SPIE. 2003, pp. 919–925.

[41] Ed Roseboom et al. “Automated full-chip hotspot detection and removal flow for interconnect layers of cell-based designs”. In: *Design for Manufacturability through Design-Process Integration*. Vol. 6521. SPIE. 2007, pp. 120–128.

[42] Andrew B Kahng, Chul-Hong Park, and Xu Xu. “Fast dual graph-based hotspot detection”. In: *Photomask Technology 2006*. Vol. 6349. SPIE. 2006, pp. 125–132.

[43] Hailong Yao et al. “Efficient process-hotspot detection using range pattern matching”. In: *Proceedings of the 2006 IEEE/ACM international conference on Computer-aided design*. 2006, pp. 625–632.

[44] Yen-Ting Yu et al. “Accurate process-hotspot detection using critical design rule extraction”. In: *Proceedings of the 49th Annual Design Automation Conference*. 2012, pp. 1167–1172.

[45] Wan-Yu Wen et al. “A fuzzy-matching model with grid reduction for lithography hotspot detection”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 33.11 (2014), pp. 1671–1680.

[46] Wan-Yu Wen et al. “A Fuzzy-Matching Model With Grid Reduction for Lithography Hotspot Detection”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 33.11 (2014), pp. 1671–1680.

[47] Sean Shang-En Tseng et al. “Efficient search of layout hotspot patterns for matching SEM images using multilevel pixelation”. In: *Optical Microlithography XXXII*. Vol. 10961. SPIE. 2019, pp. 51–58.

[48] Dragoljub Gagi Drmanac, Frank Liu, and Li-C Wang. "Predicting variability in nanoscale lithography processes". In: *Proceedings of the 46th Annual Design Automation Conference*. 2009, pp. 545–550.

[49] Duo Ding, J Andres Torres, and David Z Pan. "High performance lithography hotspot detection with successively refined pattern identifications and machine learning". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 30.11 (2011), pp. 1621–1634.

[50] Duo Ding et al. "EPIC: Efficient prediction of IC manufacturing hotspots with a unified meta-classification formulation". In: *17th Asia and South Pacific Design Automation Conference*. IEEE. 2012, pp. 263–270.

[51] Tetsuaki Matsunawa et al. "A new lithography hotspot detection framework based on AdaBoost classifier and simplified feature extraction". In: *Design-Process-Technology Co-optimization for Manufacturability IX*. Vol. 9427. SPIE. 2015, pp. 201–211.

[52] Yen-Ting Yu et al. "Machine-learning-based hotspot detection using topological classification and critical feature extraction". In: *Proceedings of the 50th annual design automation conference*. 2013, pp. 1–6.

[53] Bei Yu et al. "Accurate lithography hotspot detection based on principal component analysis-support vector machine classifier with hierarchical data clustering". In: *Journal of Micro/Nanolithography, MEMS, and MOEMS* 14.1 (2015), pp. 011003–011003.

[54] Tetsuaki Matsunawa, Bei Yu, and David Z Pan. "Laplacian eigenmaps- and bayesian clustering-based layout pattern sampling and its applications to hotspot detection and optical proximity correction". In: *Journal of Micro/Nanolithography, MEMS, and MOEMS* 15.4 (2016), pp. 043504–043504.

[55] Wei Ye et al. “Litho-GPA: Gaussian process assurance for lithography hotspot detection”. In: *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2019, pp. 54–59.

[56] Haoyu Yang et al. “Layout Hotspot Detection with Feature Tensor Generation and Deep Biased Learning”. In: *ACM/IEEE Design Automation Conference (DAC)*. 2017, 62:1–62:6.

[57] H.Y. Yang et al. “Bridging the Gap Between Layout Pattern Sampling and Hotspot Detection via Batch Active Learning”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* PP (Aug. 2020), pp. 1–1. DOI: 10.1109/TCAD.2020.3015903.

[58] Hao Geng et al. “Hotspot Detection via Attention-based Deep Layout Metric Learning”. In: *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 2020, pp. 1–8.

[59] Ran Chen et al. “Faster Region-based Hotspot Detection”. In: *ACM/IEEE Design Automation Conference (DAC)*. 2019, 146:1–146:6.

[60] Yiyang Jiang et al. “Efficient layout hotspot detection via binarized residual neural network ensemble”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 40.7 (2020), pp. 1476–1488.

[61] Shuyuan Sun et al. “Efficient hotspot detection via graph neural network”. In: *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2022, pp. 1233–1238.

[62] *KLayout*. URL: https://www.klayout.de/doc/manual/drc.html.

[63] *LayoutEditor*. URL: https://www.layouteditor.org/layoutscript/api/drc.

[64] Yibo Lin et al. “DREAMPlace: Deep Learning Toolkit-Enabled GPU Acceleration for Modern VLSI Placement”. In: *ACM/IEEE Design Automation Conference (DAC)*. 2019, pp. 1–6.

[65] Siting Liu et al. “Global Placement with Deep Learning-Enabled Explicit Routability Optimization”. In: *IEEE/ACM Proceedings Design, Automation and Test in Eurpoe (DATE)*. 2021, pp. 1821–1824.

[66] Zhiyao Xie et al. “RouteNet: Routability prediction for mixed-size designs using convolutional neural network”. In: *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 2018, pp. 1–8.

[67] Daijoon Hyun, Yuepeng Fan, and Youngsoo Shin. “Accurate wirelength prediction for placement-aware synthesis through machine learning”. In: *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2019, pp. 324–327.

[68] Yuzhe Ma et al. “A Unified Framework for Simultaneous Layout Decomposition and Mask Optimization”. In: *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 2017, pp. 81–88.

[69] Guojin Chen et al. “DAMO: Deep Agile Mask Optimization for Full Chip Scale”. In: *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 2020, pp. 1–9.

[70] Hao Geng et al. “SRAF Insertion via Supervised Dictionary Learning”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*. 2020, pp. 2849–2859.

[71] Haoyu Yang et al. “Layout hotspot detection with feature tensor generation and deep biased learning”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*. 2019, pp. 1175–1187.

[72] Ran Chen et al. “Faster region-based hotspot detection”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*. 2021, pp. 669–680.

[73] Yiyang Jiang et al. “Efficient Layout Hotspot Detection via Binarized Residual Neural Network”. In: *2019 56th ACM/IEEE Design Automation Conference (DAC)*. 2019, pp. 1–6.

[74] Christopher B Harris and Ian G Harris. “Glast: Learning formal grammars to translate natural language specifications into hardware assertions”. In: *IEEE/ACM Proceedings Design, Automation and Test in Eurpoe (DATE)*. 2016, pp. 966–971.

[75] Junchen Zhao and Ian G Harris. “Automatic assertion generation from natural language specifications using subtree analysis”. In: *IEEE/ACM Proceedings Design, Automation and Test in Eurpoe (DATE)*. 2019, pp. 598–601.

[76] Rahul Krishnamurthy and Michael S Hsiao. “Transforming natural language specifications to logical forms for hardware verification”. In: *2020 IEEE 38th International Conference on Computer Design (ICCD)*. 2020, pp. 393–396.

[77] Binwu Zhu et al. “Efficient Design Rule Checking Script Generation via Key Information Extraction”. In: *2022 ACM/IEEE 4th Workshop on Machine Learning for CAD (MLCAD)*. 2022, pp. 77–82. DOI: 10.1109/MLCAD55463.2022.9900085.

[78] Ashish Vaswani et al. “Attention is all you need”. In: *Annual Conference on Neural Information Processing Systems (NIPS)*. 2017, pp. 5998–6008.

[79] Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understand-

ing". In: *Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*. 2019, pp. 4171–4186.

[80] Peter J Liu et al. "Generating Wikipedia by Summarizing Long Sequences". In: *International Conference on Learning Representations (ICLR)*. 2018.

[81] Daniel Cer et al. "Universal sentence encoder". In: *arXiv preprint arXiv:1803.11175* (2018).

[82] Wei Yang et al. "End-to-End Open-Domain Question Answering with BERTserini". In: *Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*. 2019.

[83] Jinhua Zhu et al. "Incorporating BERT into Neural Machine Translation". In: *International Conference on Learning Representations (ICLR)*. 2019.

[84] Yukun Zhu et al. "Aligning books and movies: Towards story-like visual explanations by watching movies and reading books". In: *IEEE International Conference on Computer Vision (ICCV)*. 2015, pp. 19–27.

[85] Kirti Bhanushali. "Design Rule Development for FreePDK15: An Open Source Predictive Process Design Kit for 15nm FinFET Devices". PhD thesis. May 2014.

[86] Paul R Kingsbury and Martha Palmer. "From TreeBank to PropBank." In: *Language Resources and Evaluation Conference (LREC)*. 2002, pp. 1989–1993.

[87] Collin F Baker, Charles J Fillmore, and John B Lowe. "The Berkeley framenet project". In: *Annual Meeting of the Association for Computational Linguistics (ACL)*. 1998, pp. 86–90.

[88] *QuillBot*. URL: https://quillbot.com.

[89] John Lafferty, Andrew McCallum, and Fernando CN Pereira. "Conditional random fields: Probabilistic models for segmenting and labeling sequence data". In: *International Conference on Machine Learning (ICML)*. 2001.

[90] Silvaco. *Guardian*. URL: https://silvaco.com/wp-content/uploads/product/pdf/guardian_brief.pdf.

[91] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[92] Adam Paszke et al. "Pytorch: An imperative style, high-performance deep learning library". In: *arXiv preprint arXiv:1912.01703* (2019).

[93] Lawrence T Clark et al. "ASAP7: A 7-nm finFET predictive process design kit". In: *Microelectronics Journal* 53 (2016), pp. 105–115.

[94] Jie Zhou and Wei Xu. "End-to-end learning of semantic role labeling using recurrent neural networks". In: *Annual Meeting of the Association for Computational Linguistics (ACL)*. 2015, pp. 1127–1137.

[95] Matthew E Peters et al. "Deep contextualized word representations". In: *Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*. 2018, pp. 2227–2237.

[96] Karol Gregor and Yann LeCun. "Learning fast approximations of sparse coding". In: *International Conference on Machine Learning (ICML)*. 2010.

[97] Tianlong Chen et al. "Learning to optimize: A primer and a benchmark". In: *arXiv preprint arXiv:2103.12828* (2021).

[98] Huy Vu, Gene Cheung, and Yonina C Eldar. "Unrolling of deep graph total variation for image denoising". In: *ICASSP 2021-2021 IEEE International*

Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE. 2021, pp. 2050–2054.

[99] Vishal Monga, Yuelong Li, and Yonina C Eldar. "Algorithm unrolling: Interpretable, efficient deep learning for signal and image processing". In: *IEEE Signal Processing Magazine* 38.2 (2021), pp. 18–44.

[100] Siheng Chen, Yonina C Eldar, and Lingxiao Zhao. "Graph unrolling networks: Interpretable neural networks for graph signal denoising". In: *IEEE Transactions on Signal Processing* 69 (2021), pp. 3699–3713.

[101] Harold Horace Hopkins. "The concept of partial coherence in optics". In: *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences* 208.1093 (1951), pp. 263–277.

[102] Shayak Banerjee, Zhuo Li, and Sani R. Nassif. "ICCAD-2013 CAD contest in mask optimization and benchmark suite". In: *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 2013.

[103] Wenqian Zhao et al. "AdaOPC: A self-adaptive mask optimization framework for real design patterns". In: *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*. 2022, pp. 1–9.

[104] Adam Paszke et al. "Pytorch: An imperative style, high-performance deep learning library". In: *arXiv preprint arXiv:1912.01703* (2019).

[105] Haoyu Yang et al. "Generic lithography modeling with dual-band optics-inspired neural networks". In: *Proceedings of the 59th ACM/IEEE Design Automation Conference*. 2022, pp. 973–978.

[106] Qijing Wang et al. "A2-ILT: GPU Accelerated ILT with Spatial Attention Mechanism". In: *ACM/IEEE Design Automation Conference (DAC)*. 2022.

[107] Dragoljub G. Drmanac, Frank Liu, and Li-C. Wang. “Predicting variability in nanoscale lithography processes”. In: *ACM/IEEE Design Automation Conference (DAC)*. 2009, pp. 545–550.

[108] Duo Ding, J. Andres Torres, and David Z. Pan. “High Performance Lithography Hotspot Detection with Successively Refined Pattern Identifications and Machine Learning”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 30.11 (2011), pp. 1621–1634.

[109] Yen-Ting Yu et al. “Machine-learning-based hotspot detection using topological classification and critical feature extraction”. In: *ACM/IEEE Design Automation Conference (DAC)*. 2013, pp. 671–676.

[110] Hao Geng et al. “Sparse VLSI Layout Feature Extraction: A Dictionary Learning Approach”. In: *IEEE Annual Symposium on VLSI (ISVLSI)*. 2018, pp. 488–493.

[111] Bei Yu et al. “Accurate lithography hotspot detection based on principal component analysis-support vector machine classifier with hierarchical data clustering”. In: *Journal of Micro/Nanolithography, MEMS, and MOEMS (JM3)* 14.1 (2015), p. 011003.

[112] Hang Zhang, Bei Yu, and Evangeline F. Y. Young. “Enabling Online Learning in Lithography Hotspot Detection with Information-Theoretic Feature Optimization”. In: *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 2016, 47:1–47:8.

[113] Haoyu Yang et al. “Imbalance aware lithography hotspot detection: a deep learning approach”. In: *Journal of Micro/Nanolithography, MEMS, and MOEMS (JM3)* 16.3 (2017), p. 033504.

[114] Haoyu Yang et al. “Detecting multi-layer layout hotspots with adaptive squish patterns”. In: *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*. 2019, pp. 299–304.

[115] Yiyang Jiang et al. “Efficient Layout Hotspot Detection via Binarized Residual Neural Network”. In: *ACM/IEEE Design Automation Conference (DAC)*. 2019, 147:1–147:6.

[116] Hei Law and Jia Deng. “Cornernet: Detecting objects as paired keypoints”. In: *European Conference on Computer Vision (ECCV)*. 2018, pp. 734–750.

[117] Xingyi Zhou, Dequan Wang, and Philipp Krähenbühl. “Objects as points”. In: *arXiv preprint arXiv:1904.07850* (2019).

[118] Ashish Vaswani et al. “Attention is all you need”. In: *arXiv preprint arXiv:1706.03762* (2017).

[119] Kaiming He et al. “Deep residual learning for image recognition”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778.

[120] Shaoqing Ren et al. “Faster R-CNN: towards real-time object detection with region proposal networks”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 39.6 (2016), pp. 1137–1149.

[121] Jifeng Dai et al. “R-fcn: Object detection via region-based fully convolutional networks”. In: *arXiv preprint arXiv:1605.06409* (2016).

[122] Tsung-Yi Lin et al. “Feature pyramid networks for object detection”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 2117–2125.

[123] Tsung-Yi Lin et al. “Focal Loss for Dense Object Detection”. In: *IEEE International Conference on Computer Vision (ICCV)* (2017), pp. 2999–3007.

[124] Rasit O. Topaloglu. “ICCAD-2016 CAD contest in pattern classification for integrated circuit design space analysis and benchmark suite”. In: *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 2016, 41:1–41:4.

[125] Rayleigh. “XXXI. Investigations in optics, with special reference to the spectroscope”. In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 8.49 (1879), pp. 261–274.

[126] Jhih-Rong Gao et al. “MOSAIC: Mask optimizing solution with process window aware inverse correction”. In: *ACM/IEEE Design Automation Conference (DAC)*. 2014.

[127] Vivek Bakshi. “EUV lithography”. In: (2009).

[128] Yuzhe Ma et al. “A unified framework for simultaneous layout decomposition and mask optimization”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39.12 (2020), pp. 5069–5082.

[129] Andres J. Torres. “ICCAD-2012 CAD Contest in Fuzzy Pattern Matching for Physical Verification and Benchmark Suite”. In: *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 2012, pp. 349–350.

[130] Kaiming He et al. “Masked autoencoders are scalable vision learners”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2022, pp. 16000–16009.

[131] Alec Radford et al. “Improving language understanding by generative pre-training”. In: (2018).

[132] Zekai Chen et al. “Masked image modeling advances 3d medical image analysis”. In: *IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*. 2023, pp. 1970–1980.

[133] Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. In: *Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*. 2019, pp. 4171–4186.

[134] Zhenda Xie et al. “Simmim: A simple framework for masked image modeling”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2022, pp. 9653–9663.

[135] Alexey Dosovitskiy et al. “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale”. In: *International Conference on Learning Representations (ICLR)*.

[136] Ze Liu et al. “Swin transformer: Hierarchical vision transformer using shifted windows”. In: *IEEE International Conference on Computer Vision (ICCV)*. 2021, pp. 10012–10022.

[137] Tsung-Yi Lin et al. “Feature pyramid networks for object detection”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 2117–2125.

[138] Hengshuang Zhao et al. “Pyramid scene parsing network”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 2881–2890.

[139] Su Zheng et al. “LithoBench: Benchmarking AI Computational Lithography for Semiconductor Manufacturing”. In: *Advances in Neural Information Processing Systems* 36 (2024).

[140] Guojin Chen et al. “DevelSet: Deep neural level set for instant mask optimization”. In: *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 2021.

[141] Binwu Zhu et al. “Hotspot detection via multi-task learning and transformer encoder”. In: *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE. 2021, pp. 1–8.

[142] Ziyang Yu et al. “CTM-SRAF: Continuous Transmission Mask-based Constraint-aware Sub Resolution Assist Feature Generation”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* (2023).