LLM-HD: Layout Language Model for Hotspot Detection with GDS Semantic Encoding

Yuyang Chen*
ShanghaiTech University

Yiwen Wu*
ShanghaiTech University

Jingya Wang
ShanghaiTech University

Tao Wu
ShanghaiTech University

Xuming He
ShanghaiTech University

Jingyi Yu†
ShanghaiTech University

Hao Geng†
ShanghaiTech University
Zhangjiang Laboratory

Abstract

Layout hotspot detection approaches are challenged by the time-to-market constraint and complex designs under rapid downscaling of technology nodes. Pattern matching and learning-based detectors are proposed as quick detection methods. These layout image-based detectors use images transformed from binary database files of layout like GDSII as their inputs. Italy leads to foreground information (e.g., metal polygons) loss and even distortion when shrinking the image size to fit the approach input. Moreover, plenty of irrelevant background information such as non-polygon pixels is also fed into the model, which hinders the fitting of the model and results in a waste of computational resources. In this work, for the first time, we propose a new layout hotspot detection paradigm, where hotspots are directly detected on binary database files by exploiting a hierarchical GDS semantic representation scheme and a well-designed pre-trained natural language processing (NLP) model. Compared with state-of-the-art (SOTA) works, the proposed detector achieves better results on both the `ICCAD2012` metal layer benchmark and the more challenging `ICCAD2020` via layer benchmark, which demonstrates the effectiveness and efficiency.

1 Introduction

The continuous shrinkage of technology nodes poses great challenges to many aspects of industrial flow, such as pitch reduction, patterning flexibility, and lithography processing variability. As a result, the layout is more susceptible to having defects after fabrication, which leads to substantial yield loss and elongates the design-to-manufacturing cycle. Early detection of such defects is crucial, prompting the need for innovative paradigms to identify potential lithography hotspots, which are lithographical-sensitive/defect layout patterns.

Traditional methods to ensure hotspot-free designs at tape-out, such as lithography simulation and pattern matching [1, 2], have their own limitations. Although lithography simulation is accurate, it is computation-intensive. Pattern matching, while faster, fails to detect unseen patterns. Machine learning-based detector [3] offers a promising alternative, yet suffers the trade-off between accuracy and false alarm. With the deep learning techniques including convolutional neural networks (CNNs) pervading our community, deep neural network-based detection approaches [4, 5] have been proposed. Nevertheless, these methods cannot explore the relationships between long-distance patterns in a large-sized layout, which neglects the nature of light propagation of advanced lithography. Besides, due to the coarse design, the detection ability of these detectors is not expected, which is revealed by detecting on a more challenging via layer benchmark, `ICCAD2020` proposed in [6]. Recent approaches utilizing attention mechanism [6] and graph neural networks (GNNs) [7] attempt to tackle the issues and enhance the detection performance.

The layout patterns are typically stored in the binary file format like GDSII [8] or OASIS [9] in the integrated circuits (ICs) industry. In data pre-processing, prior methods primarily focus on

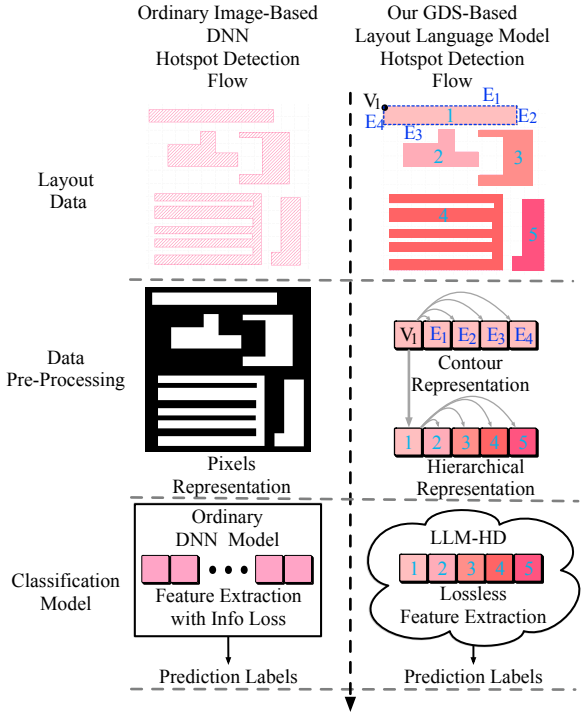


Figure 1: Illustration of our layout language model with GDS-based lossless encoder concerning the ordinary Image-based DNN models.

*Both authors contributed equally to this research.
†Corresponding author

DAC '24, June 23–27, 2024, San Francisco, CA, USA

ACM ISBN 979-8-4007-0601-1/24/06.
https://doi.org/10.1145/3649329.3658479

transforming Graphic Design System (GDS) files into pixel-based images for further analyzing or extracting layout features. These image-centric approaches, despite their prevalence, have disadvantages. Such image-based layouts introduce unnecessary redundancy and even hinder the fitting of the learning model since hotspots are primarily defined by spatial polygon configurations, not all pixelated layout data. Even worse, the layout images may be distorted when they are downsampled to fit the input size of deep neural network-based detectors. Inefficient layout representation and error-producing downsampling lead to feature extracting with information loss. As a result, learning models suffer a major performance degradation. An illustration of the ordinary deep neural network (DNN) model-based hotspot detection flow is shown in Figure 1. These issues highlight the need for a more data-faithful representation method for effective hotspot detection.

Recent advancements in Large Language Models (LLMs), particularly OpenAI's GPT-n series [10], have shown remarkable capabilities in encoding and applying textual knowledge, resulting in superior performance on tasks requiring logical inference. Motivated by the great success of LLMs and to tackle the data-faithful representation issue for hotspot detection, in this paper, we propose LLM-HD, a layout Language model for hotspot detection with GDS semantic encoding. To the best of our knowledge, this is the first work to perform hotspot detection on binary database files of layout applying the language model. The illustration of our framework is shown in Figure 1. By applying the language model with sentiment analysis, we semantize the hotspot detection problem in the field of natural language processing (NLP). As illustrated in Figure 2, our layout language model has an outstanding feature mapping ability on the layout patterns. Both the feature mapping of metal layout patterns in (a) (b) and via layout patterns in (c) (d) are well-distributed.

The main contributions of this paper are as follows:

- We formulate layout hotspot detection as a semantic analysis task and apply LLM techniques to process the data, thereby extending the application scope of language models within the field of semiconductor manufacturing.
- We introduce a GDS-based hierarchical semantic encoding mechanism for layout patterns that is lossless, time-efficient, and resource-saving.
- Our method outperforms state-of-the-art (SOTA) works by achieving 2% accuracy improvement and 18% false alarms reduction on benchmarks.

2 Preliminaries

In this section, we first give some basic concepts about multi-head self-attention which is the foundation of our layout language model. Then, given the necessary definition of evaluation metrics, the problem formulation is provided.

2.1 Multi-Head Self-Attention

Transformers act as the backbone of state-of-the-art models like BERT [11] and GPT [10] in natural language processing. Multi-head self-attention (**MHSA**) is a basic element of the transformer. For primitive self-attention, the input is a sequence $\mathbf{X} \in \mathbb{R}^{n \times c}$, where n is the number of encoded entities in terms of global information, c is the number of embedded channels. Three weight matrices $\mathbf{W}_Q \in \mathbb{R}^{c \times d_q}$, $\mathbf{W}_K \in \mathbb{R}^{c \times d_k}$, and $\mathbf{W}_V \in \mathbb{R}^{c \times d_v}$ are learned, with $d_q = d_k$. Then

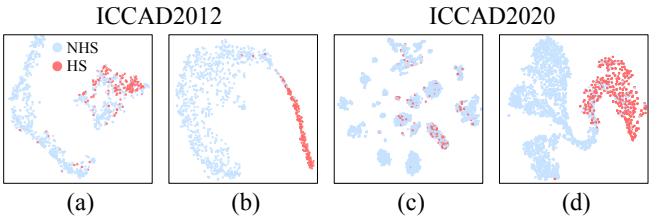


Figure 2: Illustration of our layout language model's feature mapping ability. Both (a) and (c) are the feature mappings of layout patterns before training. Both (b) and (d) are the feature mapping of layout patterns after training.

the sequence $\mathbf{X}$ interact with weighted matrices, *i.e.*, $\mathbf{Q} = \mathbf{X}\mathbf{W}_Q$, $\mathbf{K_c} = \mathbf{X}\mathbf{W}_K$, $\mathbf{V_c} = \mathbf{X}\mathbf{W}_V$. The **MHSA** block with H heads can be calculated as follows:

$$\mathbf{Z}_h = \mathbf{softmax}(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_q}})\mathbf{V}, h = 0, \cdots, H-1, \tag{1}$$

$$\mathbf{Z} = \mathbf{concat}(\mathbf{Z}_0, \cdots, \mathbf{Z}_{H-1})\mathbf{W}_o, \tag{2}$$

where $\mathbf{W}_o \in \mathbb{R}^{H \cdot d_v \times c}$ is the projection weight. Adding L **mhsa** blocks, the whole module can be shown as follows:

$$\mathbf{Z}'_{(l)} = \mathbf{mhsa}(\mathbf{ln}(\mathbf{Z}_{(l-1)})) + \mathbf{Z}_{(l-1)}, \tag{3}$$

$$\mathbf{Z}_{(l)} = \mathbf{mlp}(\mathbf{ln}(\mathbf{Z}'_l)) + \mathbf{Z}'_{(l)}, \tag{4}$$

where **ln** is layer normalization. The addition of the formula Equation (3) and Equation (4) aims to make the network residential. The illustration of **MHSA** is shown in Figure 5.

2.2 Problem Formulation

The key metrics used for the detector's performance evaluation are listed as follows:

Definition 1 (Accuracy)**.** The ratio of the hotspot clips that are correctly detected to the whole number of real hotspot clips.

Definition 2 (False Alarm)**.** The number of non-hotspot clips that are incorrectly judged as hotspots.

The specified problem concerning the above metrics is as follows:

Problem 1 (Language Model-based Layout Hotspot Detection)**.** Given layout clips in binary file format like GDSII, our goal is to harness language model-based detectors to perform hotspot detection, maximizing the accuracy and minimizing the false alarm.

3 Proposed Framework

In this section, we present the whole architecture of our framework. First, we introduce our direct GDS-based layout pattern preprocessing, which introduces our insights and explains our semantic layout encoding method in detail concerning the traditional layout representation method. Secondly, the architecture of the proposed layout language model is presented. Finally, the calibration flow and testing of the LLM-HD are highlighted, where the calibration flow is divided into the pre-training stage and the fine-tuning stage.

3.1 Efficient GDS-Based Layout Pre-Processing

3.1.1 Insights As claimed in Section 2, the formation of hotspots is caused by certain spatial arrangements of layout patterns, related to both the relative spatial permutation of different polygons and the

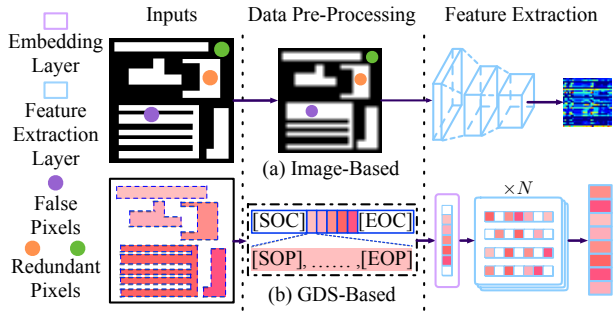


Figure 3: This is the layout processing flow. (a) is the image-based layout processing flow. (b) is our GDS-based layout processing flow.

configuration of each polygon. This means that polygons' contours combined with relative spatial permutation are the key features of layout patterns. Original layout patterns are in the form of GDS files during the fabrication of ICs. In GDS files, the polygons of layout patterns are netlist shaping, representing the contours of polygons with coordinates. For traditional hotspot detection missions, GDS-based layout patterns are first transformed into image files. This transformation leads to the redundancy of layout representation, which is shown in Figure 3. As we only need the representation of polygons' contours combined with spatial permutation, pixels inside the polygons and background pixels far from polygons are redundant pixels for the model to detect the hotspots. Also, in most cases, the image-based layout patterns are in huge sizes considering the input of the model. They are usually downsampled into suitable sizes to do the input feature extraction, which is also shown in Figure 3. This can easily lead to false representation for some polygons close enough can be merged, most likely making a non-hotspot pattern into a hotspot pattern. We directly use GDS-based layout patterns and encode them hierarchically and semantically, which is more efficient. We represent layout patterns considering the shape of each polygon and the relative permutation of different polygons. By pre-processing the layout patterns in this way, we efficiently represent the layout patterns with no information loss.

3.1.2 Squish Patterns Representation The squish patterns [12] compress the layout images into a topology matrix $\mathbf{T}$ and two geometric vectors $\delta_{\mathbf{x}}$ and $\delta_{\mathbf{y}}$, shown in Figure 4. Topology matrix $\mathbf{T}$ represents the existence of the polygons in the defined area, where 1 is a yes and 0 otherwise. Geometric vectors $\delta_{\mathbf{x}}$ and $\delta_{\mathbf{y}}$ represent the widths and heights of each grid in x and y axises. This representation of layout patterns is more efficient than the image. However, it faces challenges for much larger and more complex layout patterns which beyond the capacity of the model. Also, the diversity and flexibility of squished layout patterns is limited.

3.1.3 Hierarchical Semantic Sequential Encoding It is quite challenging for NLP models to understand the non-semantic inputs. 2D spatial representation of layout patterns should also be encoded into 1D sequential inputs for language models without information loss. We semantically encode layout patterns into hierarchical sequential inputs without information loss with respect to [13]. The representation method is shown in Figure 4.

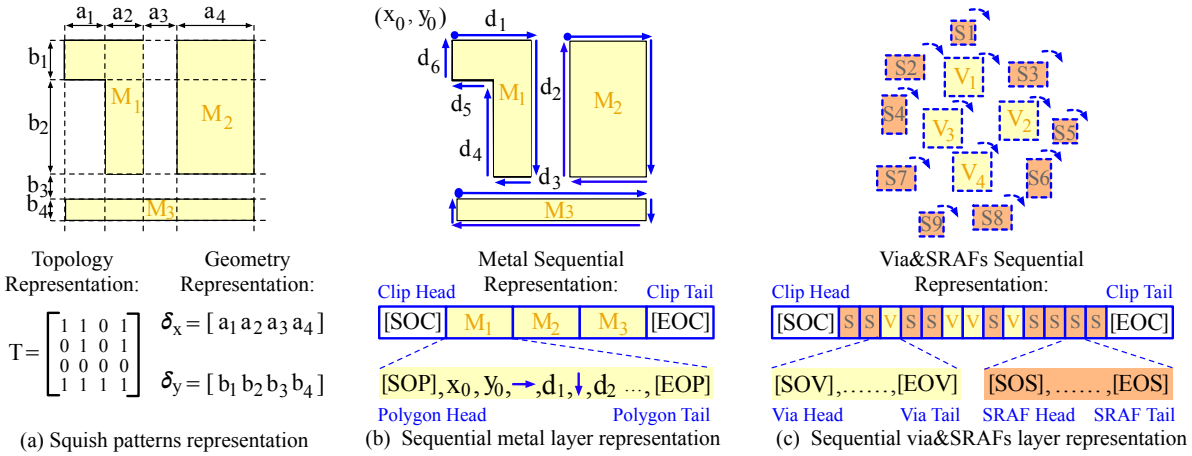
For a basic layout pattern, the representation is organized hierarchically. The structure of the representation is organized in two tiers. The outer structure starts with $[SOC]$ and ends up with $[EOC]$, which denotes the head and tail of each layout clip. The structure consists of arranged polygons inside the layout clip, organized from left to right, up to bottom. Each polygon is represented as a sequence containing the head and tail.

For different kinds of datasets, we use different encoding strategies. The illustration is shown in Figure 4. For the metal layer-based benchmark, we set the head and tail of each polygon sequence $[SOP]$ and $[EOP]$. To make the spatial relationship of the layout clip semantic, we use the starting point and clockwise traverse contour. The first two elements are coordinates of the starting point, which is the upleft corner of the polygon. Then we use the language-based directions of "up", "down", "left", and "right", followed by the distance concerning the next point to describe the contour. As for the via-layer-based benchmark, we encode the layout patterns using different separations by the functionality. Each layout clip consists of two kinds of patterns. One is via pattern, which is the pattern to be etched on the wafer. The other is the sub-resolution assist features (SRAFs) pattern, which is used to assist the lithography system to etch via pattern. SRAFs shouldn't be etched on the wafer. The head and tail of via patterns are $[SOV]$ and $[EOV]$. The head and tail of SRAFs are $[SOS]$ and $[EOS]$. It should be noticed that the coordinates of the starting points for all the patterns are relative to the first polygon's starting point. We utilized the semantic layout encoding method on layout patterns and considered different kinds of encoding on different layout patterns. The method is more flexible than squish patterns representation and there is no information loss during the whole encoding process.

3.2 Architecture of Layout Language Model

Standard conditional language models can only be trained in one direction, either from left to right or right to left. The only usage of a single direction of the sequential layout patterns representation relationship leads to inefficiency of language model learning [11]. The representation of a certain part of the layout clip not only depends on the past representation before but also depends on the future representation after concerning the time frame. Bidirectional Encoder Representation from Transformers (BERT) [11] is a fully connected self-attention model that can model the relation of patterns description words and let the model learn the structure information of layout patterns. We take advantage of BERT and design our layout language model for hotspot detection. By absorbing the directional representation of sequential layout patterns forward and backward, our model can better capture the meaning of each representation element in sequential layout pattern inputs, understanding the relationship between polygons both locally and globally.

The illustration of our layout language model architecture is shown in Figure 5. Sequential layout inputs first go through the Embedding layer, then the stacking LLM-HD layers, and finally the output layer. The embedding layer embeds the sequential layout inputs into tokens, with label information and positional information. The LLM-HD layer is the key layer in the feature extraction process of our model. It possesses the **MHSA**, **Add** & **Norm** layer, and **Feed Forward** layer. The **MHSA** is introduced in Section 2.1, learning the relationship of pattern description words and finding the affected spatial arrangement that leads to hotspot patterns. The **Add** & **Norm** layer is the combination of the residential layer and



(a) Squish patterns representation (b) Sequential metal layer representation (c) Sequential via&SRAFs layer representation

Figure 4: Squish patterns representation vs our hierarchical semantic encoding.

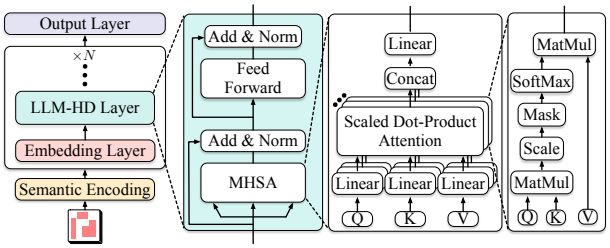


Figure 5: Illustration of our language model's architecture.

normalization layer, used to keep the distribution of inputs during training. The **Feed Forward** layer is a linear layer that deepens the network. As for the output layer, it is a linear layer with an activation function, the size of which is 2, which corresponds to the hotspot detection mission. By using such a powerful structure, our LLM-HD easily detects the long-range dependency of sequential layout patterns, which is reflected in our results.

3.3 Flow of LLM-HD

3.3.1 Pre-training Our layout language model empowers the bidirectional layout feature representation by pre-training the model with unsupervised learning. For ordinary language-based pre-training, masked language modeling (MLM) and next sentence prediction (NSP) are used, which are the two main unsupervised tasks. To better learn the representation features of the sequential layout patterns, we chose MLM as our pre-training unsupervised task on the merged training set of `ICCAD2012` and `ICCAD2020` benchmarks. Unsupervised-learning-based pre-training stage of LLM-HD is shown in Figure 6. The GDS-based layout is encoded semantically and hierarchically and embedded into tokenized sequential inputs by the embedding layer. The unsupervised learning task doesn't need label information, thus the labels of the layout should be erased. Both the global representation and the local representation of the layout need to be reconstructed. Globally, for the sequential polygon inputs in each clip, some polygons are masked and the whole clipped layout patterns need to be reconstructed. Locally, some parts of the selected polygons are masked, and the representation of the polygons needs to be reconstructed. We randomly choose 15% of the representation in each clip sequential layout input and mask them with `[MASK]`. For the masked polygon representation, 80% time of their states is `[MASK]`, with 10% time being random tokens and 10% time being still. We only predict the masked layout representation rather than reconstructing the entire sequential layout inputs. The usage of pre-training with MLM not only makes the model learn the representation of layout patterns but also prevents the model from overfitting during the fine-tuning stage.

3.3.2 Fine-tuning It should be noticed that in the encoding session Section 3.1.3, we transfer the clipping layout patterns into sequential inputs. We straightforwardly use sequential classification as our downstream task. The sequential classification task is used to do the sentiment analysis. Sentiment analysis is used to understand human emotions and assign emotional labels to the sequential inputs. `Hotspot` and `non-hotspot` are two labels we allocated to the clipped layout patterns. We fine-tune our layout language model on each training set of `ICCAD2012` benchmark and `ICCAD2020` benchmark. The illustration of the LLM-HD fine-tuning stage is shown in Figure 6. Partially similar to the pre-training stage, the GDS layout is encoded semantically and embedded sequentially. As labels of layout patterns are necessary, the data should be labeled. The weights of the pre-training stage are transferred to the fine-tuning stage.

3.3.3 Testing We take advantage of the calibration flow of LLM-HD by consistently pre-training and fine-tuning. After fine-tuning the layout language model on the downstream sequential classification task, we tested its performance on all the testing sets. It achieves outstanding results on the benchmarks, details are shown in Section 4.

4 Experiments

We implement our work with the Pytorch-Geometric toolkit [15] and the HuggingFace package [16] in Python. We train and test our model on the platform that possesses the Xeon Gold 6226R CPU and the Nvidia Tesla A100 Graphic card. Two benchmarks act as the efficiency and effectiveness verification of our model. The whole information of these benchmarks is shown in Table 1. One is the metal-layer-based `ICCAD2012` benchmark, provided by the ICCAD contest. The other is the via-layer-based `ICCAD2020` benchmark, which is more challenging because its technique node is under 45 nm.

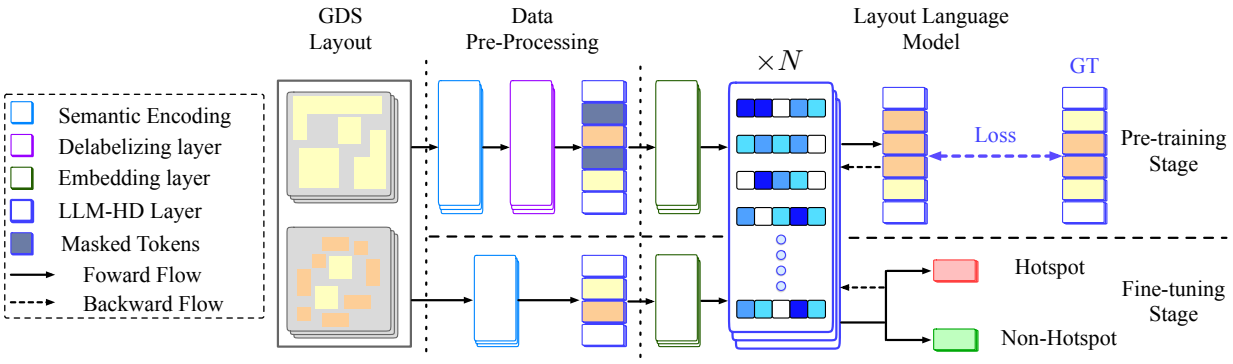


Figure 6: The calibration flow of LLM-HD.

Table 1: Benchmarks Statistics.

Benchmarks	Training Set		Testing Set		Size/Clip
	HS#	NHS#	HS#	NHS#	(μm^2)
ICCAD2012	1204	17096	2524	13503	3.6×3.6
Via-1	3418	10302	2267	6878	2.0×2.0
Via-2	1029	11319	724	7489	2.0×2.0
Via-3	614	19034	432	12614	2.0×2.0
Via-4	39	23010	26	15313	2.0×2.0
Via-Merge	5100	63665	3449	42294	2.0×2.0

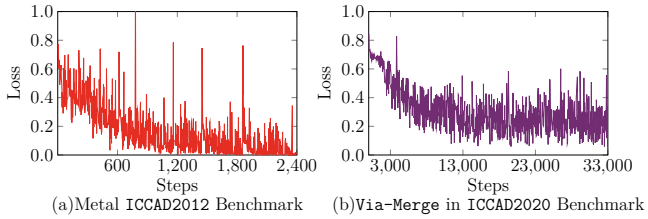
ICCAD2020 benchmark contains four individual benchmarks and one merged benchmark. Each individual benchmark in ICCAD2020 benchmark contains different densities of via patterns. Each dataset is divided into training and testing sets. The label "HS#" means hotspot pattern and "NHS#" means non-hotspot pattern.

The following is our layout language model's configuration. The network possesses 12 encoder layers, 768 for hidden sizes, and 12 attention headers. The vocabulary of the data is pre-trained and the max length of position embedding is set to 512. We use cross entropy as our loss function and GELU as our activation function. AdamW is applied as our gradient optimizer and our batch size is set to 200. Our learning rate is set empirically with 10^{-5}. The hidden layers of our model are dropped with a probability of 0.2, making the learning process robust. The application and effect of our enhancement techniques are listed and studied in detail in Section 4.3.

We compared our layout language model with the following state of the arts on the two benchmarks to evaluate its performance.

- TCAD'19 [4] A model based on feature extraction and bias learning.
- DAC'19 [5] A fast-region-based model using aggregated feature extraction and refinement.
- ICCAD'20 [6] A CNN enabled by attention.
- DATE'22 [7] A layout-graph mapping empowered Graph Neural Network (GNN).
- CLIP'21 [14] A fine-tuned large vision language model classifier.

Comparison results are listed in Table 2. It should be noted that we get the source code of all the state of the arts from the author on GitHub and they are all run on the same platform we test our framework locally.



(a)Metal ICCAD2012 Benchmark (b)Via-Merge in ICCAD2020 Benchmark

Figure 7: Training loss of the two benchmarks.

4.1 Training Details

Our training loss on the two benchmarks is presented in Figure 7. The learning process on ICCAD2012 converged promptly with 2400 steps and loss close to 0. This states the ease of feature extraction on metal benchmarks, illustrating the power of our layout language model. Meanwhile, the process on Via-Merge in ICCAD2020 is relatively harder. The loss reduces over numerous iterations and begins to vibrate after shrinking to 0.2 with 13000 steps. It is much more challenging for the layout language model to map the features of ICCAD2020 benchmark.

4.2 Results Comparison

Comparison with the state of the arts is shown in Table 2, with three metrics listed. The Accu and FA are the accuracy and false alarms, which are introduced in Section 2. Time is the inference time of the model. It should be noted that the time doesn't contain the pre-processing time of each framework.

In Table 2, the best metrics in each case are marked in bold font. Our layout language model exceeded DATE'22 with 2% on average accuracy and 18% average false alarm reduction. For the metal ICCAD2012 benchmark, although the promotion of accuracy, which is 0.5% less than the DAC'19, we achieve 188% and 53% false alarm reduction compared with DAC'19 and DATE'22. As for the via ICCAD2020 benchmark, our framework performs remarkably on the Via-Merge. With respect to DATE'22, we elevate 2.33% on accuracy and 11% false alarm reduction. This shows the effectiveness of our functional-based semantic embedding. The impact of this valid technique is explored in Section 4.3. However, our model runs slower than DATE'22, caused by the length of semantic encoding.

4.3 Ablation Studies

As illustrated in Section 3.1.3, the GDS-based layout patterns are encoded hierarchically and semantically. However, it's still quite challenging for the ordinary language model to fully understand

Table 2: Comparison with state of the arts.

Bench	TCAD'19 [4]			DAC'19 [5]			ICCAD'20 [6]			DATE'22 [7]			CLIP'21 [14]			LLM-HD (Ours)		
	Accu (%)	FA	Time (s)	Accu (%)	FA	Time (s)	Accu (%)	FA	Time (s)	Accu (%)	FA	Time (s)	Accu (%)	FA	Time (s)	Accu (%)	FA	Time (s)
ICCAD2012	98.40	3535	313.38	**98.54**	3260	349.90	98.42	2481	89.64	98.42	1732	2.21	94.69	5005	352.69	98.49	1132	9.87
Via-1	71.50	773	27.03	89.85	1886	36.01	93.42	1589	12.36	95.41	1722	1.33	89.32	2570	24.18	**95.76**	1702	5.73
Via-2	65.06	1290	24.95	73.00	1222	13.50	86.32	1100	8.24	90.33	1507	0.90	82.04	1355	21.45	**91.30**	1160	5.26
Via-3	48.15	760	37.55	73.38	3406	26.90	88.20	2105	12.90	89.81	1928	1.12	67.13	1801	34.22	**91.43**	1637	8.40
Via-4	76.92	155	42.04	73.08	15288	32.40	80.77	152	12.89	84.62	400	1.26	65.38	2150	39.58	**92.31**	304	9.62
Via-Merge	88.01	7633	103.39	90.42	9295	65.64	92.20	6453	37.24	93.33	5502	3.28	86.29	7022	117.30	**94.66**	4941	29.44
Average	74.67	2357.67	91.39	83.06	5726.17	87.39	89.89	2313.33	28.88	91.99	2131.67	**1.68**	80.81	3317.17	98.19	**93.99**	**1812.67**	11.39
Ratio	0.79	1.30	8.02	0.88	3.16	7.67	0.96	1.28	2.54	0.98	1.18	**0.15**	0.86	1.83	8.62	**1.00**	**1.00**	1.00

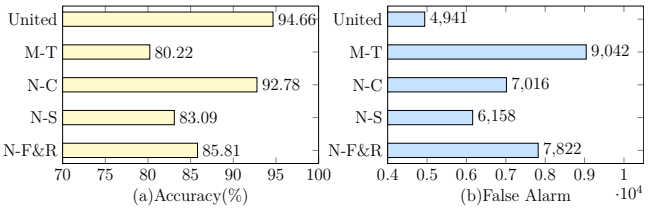


Figure 8: Ablation studies on `Via-Merge` for different techniques' application.

the semantic information. We use several augmentation methods to enrich the representation of layout patterns. For the layout patterns, the operation of rotate 180° and flip-flop doesn't change its lithographic nature. Considering the hardness of learning the spatial features of the hotspot patterns for the layout language model, we only do the rotation and flip-flop augmentation on the hotspot of the training dataset. Also, as our representation of layout patterns uses relative spatial relationships, it is not quite efficient for our model to learn the features robustly. Local shifting on the layout patterns is applied to the whole training dataset to make the process smooth. The shifting of the layout patterns is done randomly with a range from −10 to 10. Tokens redundancy is also one issue we encounter in our layout language model. We crop the layout patterns to 80% to make the semantic encoding more efficient. From Section 3.1.3, different hierarchical head and tail is applied to `ICCAD2020` benchmark to handle different functional patterns. To test the efficiency of the techniques we mentioned above, we verify in the `Via-Merge` in the `ICCAD2020` benchmark. The results are shown in Figure 8. M-T, N-C, N-S, N-F&R, and United are not different separation tokens, not using cropping, not using shifting, not using flip-flops with rotation, and usage of all techniques together. From Figure 8, we can list the effectiveness of these techniques on both accuracy and false alarm. For accuracy, the impact rank from high to low is different separation tokens, local shifting, flip-flops with rotation, and cropping. The accuracy factor values are 14.44%, 11.57%, 8.85% and 1.88%. On false alarm, the impact rank from high to low is different separation tokens, flip-flops with rotation, cropping, and local shifting. The false alarm factor values are 83%, 58%, 42%, and 25%. The different separation tokens technique possesses the most impact, which is one of our main contributions to hierarchical semantic encoding.

5 Conclusion

In this paper, for the first time, we have proposed a language model-based layout hotspot detector with binary database files of layout as our inputs. Compared to earlier image-based detectors, it is more practical and effective because there is a chance of information loss or distortion when converting the layout from GDSII or OASIS format to the image. Specially, hierarchical semantic encoding is proposed to cooperate with the layout language model, which contains the spatial relationship and shape information and thus enables a valid representation of the layout patterns. The layout language model is calibrated with pre-training and fine-tuning techniques and achieves promising results. Moreover, our work reveals and verifies the potential of applying the language model in hotspot detection.

Acknowledgement

This work is sponsored by Shanghai Pujiang Program (Project No. 22PJ1410400)

References

[1] W.-Y. Wen, J.-C. Li, S.-Y. Lin, J.-Y. Chen, and S.-C. Chang, "A fuzzy-matching model with grid reduction for lithography hotspot detection," *IEEE TCAD*, vol. 33, no. 11, pp. 1671–1680, 2014.
[2] Y.-T. Yu, Y.-C. Chan, S. Sinha, I. H.-R. Jiang, and C. Chiang, "Accurate process-hotspot detection using critical design rule extraction," in *Proc. DAC*, 2012.
[3] T. Matsunawa, J.-R. Gao, B. Yu, and D. Z. Pan, "A new lithography hotspot detection framework based on AdaBoost classifier and simplified feature extraction," in *Proc. SPIE*, vol. 9427, 2015.
[4] H. Yang, J. Su, Y. Zou, Y. Ma, B. Yu, and E. F. Y. Young, "Layout hotspot detection with feature tensor generation and deep biased learning," *IEEE TCAD*, vol. 38, no. 6, pp. 1175–1187, 2019.
[5] Y. Jiang, F. Yang, H. Zhu, B. Yu, D. Zhou, and X. Zeng, "Efficient layout hotspot detection via binarized residual neural network," in *Proc. DAC*, 2019.
[6] H. Geng, H. Yang, L. Zhang, J. Miao, F. Yang, X. Zeng, and B. Yu, "Hotspot detection via attention-based deep layout metric learning," in *Proc. ICCAD*, 2020.
[7] S. Sun, Y. Jiang, F. Yang, B. Yu, and X. Zeng, "Efficient hotspot detection via graph neural network," in *Proc. DATE*, 2022.
[8] K. Kuriyama, J. Hirumi, N. Yoshioka, Y. Hojo, Y. Kawase, S. Hara, M. Hoga, S. W. Watanabe, M. Inoue, H. Kawase *et al.*, "Development of new stream format with GDSII upper compatibility and high compression rate," in *Proc. SPIE*, vol. 4889, 2002.
[9] S. F. Schulze, P. LaCour, and L. Grodd, "OASIS-based data preparation flows: Progress report on containing data size explosion," in *Proc. SPIE*, vol. 5379, 2004.
[10] R. OpenAI, "Gpt-4 technical report," *CoRR*, 2023.
[11] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," *CoRR*, 2018.
[12] F. Gennari and Y. Lai, "Topology design using squish patterns," 2014.
[13] L. Wen, Y. Zhu, L. Ye, G. Chen, B. Yu, J. Liu, and C. Xu, "Layoutransformer: generating layout patterns with transformer via sequential pattern modeling," in *Proc. ICCAD*, 2022.
[14] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark *et al.*, "Learning transferable visual models from natural language supervision," in *Proc. ICML*, 2021.
[15] M. Fey and J. E. Lenssen, "Fast graph representation learning with pytorch geometric," *CoRR*, 2019.
[16] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz *et al.*, "HuggingFace's Transformers: State-of-the-art natural language processing," in *Proc. EMNLP*, 2020.