DRC-SG 2.0: Efficient Design Rule Checking Script Generation via Key Information Extraction

BINWU ZHU and XINYUN ZHANG, The Chinese University of Hong Kong
YIBO LIN, Peking University
BEI YU and MARTIN WONG, The Chinese University of Hong Kong

Design Rule Checking (DRC) is a critical step in integrated circuit design. 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 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. We further enhance the performance of the extractor with three specific design rule generation techniques and a multi-task learning-based rule classification module. Experimental results demonstrate that the framework can generate a single rule script in 5.46 ms on average, with the extractor achieving 91.1% precision and 91.8% recall on the key information extraction. Compared with the manual generation, our framework can significantly reduce the turnaround time and speed up process design closure.

CCS Concepts: • **Hardware → Methodologies for EDA;**

Additional Key Words and Phrases: Design Rule Checking, natural language processing, key information extraction

ACM Reference format:
Binwu Zhu, Xinyun Zhang, Yibo Lin, Bei Yu, and Martin Wong. 2023. DRC-SG 2.0: Efficient Design Rule Checking Script Generation via Key Information Extraction. *ACM Trans. Des. Autom. Electron. Syst.* 28, 5, Article 80 (September 2023), 18 pages.
https://doi.org/10.1145/3594666

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 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

This work is supported The Research Grants Council of Hong Kong SAR (Project No. CUHK14208021).
Authors' addresses: B. Zhu, X. Zhang, B. Yu, and M. Wong, The Chinese University of Hong Kong; emails: bwzhu@cse.cuhk.edu.hk, xyzhang21@cse.cuhk.edu.hk, byu@cse.cuhk.edu.hk, mdfwong@cuhk.edu.hk; Y. Lin, Peking University; email: yibolin@pku.edu.cn.
Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.
1084-4309/2023/09-ART80 $15.00
https://doi.org/10.1145/3594666

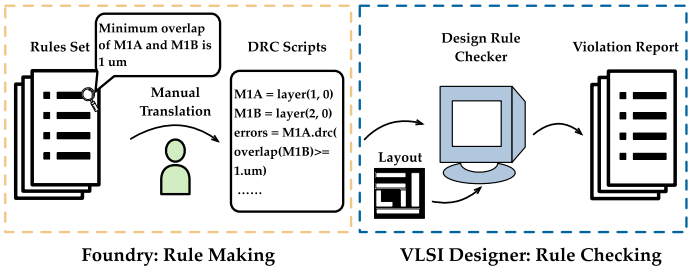


Fig. 1. Entire design rule checking flow.

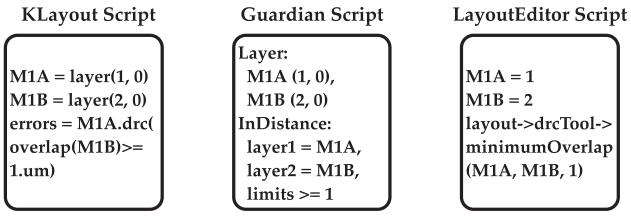


Fig. 2. Script languages vary in different checkers.

the first phase of Figure 1. (2) Rule checking: These scripts are provided to a design rule checker, such as KLayout [1] and LayoutEditor [2], 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 65-nm nodes to thousands of rules in 7-nm nodes. (2) Different checkers require different script languages, resulting in additional efforts to re-implement and transfer scripts between checkers, as shown in Figure 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 [3–6], mask synthesis [7–10], physical verification [11–14], testing [15–17], and so on. 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. [18] 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. [19] 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 turnaround 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 Reference [20], 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, Reference [20] is the first work to investigate methods for efficient DRC script generation. As shown in Figure 3, the proposed flow in Reference [20] relies on an automatic DRC script generation engine that 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

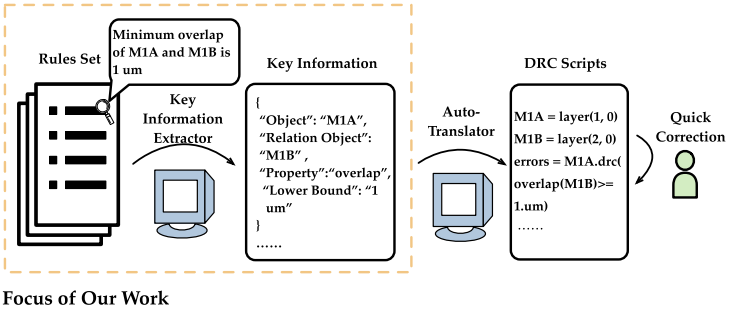


Fig. 3. Our proposed automatic DRC script generation (DRC-SG) flow. This is the optimization aiming at the rule-making phase in Figure 1.

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. 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 turnaround 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 toward the major class in the dataset. In this work, 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 7-nm technology node demonstrate that our extractor achieves 91.1% precision and 91.8% recall on the key information extraction task. Besides, it only takes 5.46 ms on average for our flow to generate the script of a single design rule.

The rest of this article is organized as follows. Section 2 introduces the problem formulation and terminologies related to this work. Section 3 illustrates the details of the preliminary work, including methods for rule data generation, and components of our proposed DRC-SG flow. Section 4 describes the techniques to enhance the performance of the preliminary DRC-SG flow. Section 5 shows the benchmarks and the experimental results, followed by the conclusion in Section 6.

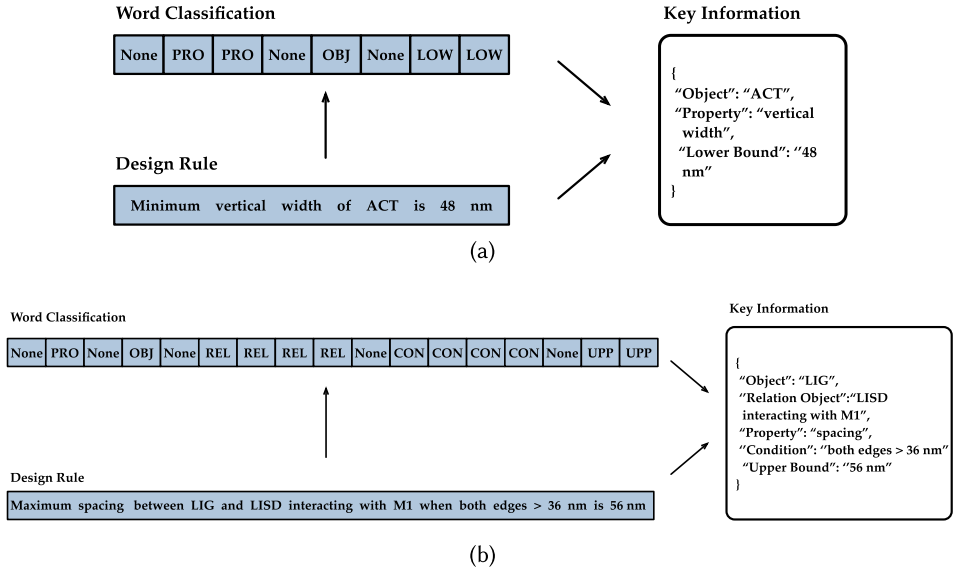


Fig. 4. Key information extraction process on (a) simple rule and (b) complex rule. (PRO means Property, OBJ means Object, LOW means Lower Bound, and CON means Condition.)

2 PRELIMINARIES

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 4(a) and Figure 4(b) 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 3.1. 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 metrics in multi-class classification problem including precision, recall, and F1 score. To illustrate these metrics clearly, we first give the confusion matrix as shown in Table 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 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 $\text{Precision} = \frac{\text{TP}}{\text{TP+FP}}$.

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

Table 1. Confusion Matrix

		Prediction	
		Positive	Negative
Actual	Positive	TP	FN
	Negative	FP	TN

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

2.2 Transformer and BERT

Recently, Transformer [21] has made much progress in sequence-to-sequence tasks [22–24]. BERT [22] 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 to solve many natural language processing problems such as Question Answering [25], Machine Translation [26], and so on. To illustrate BERT [22], we first introduce the structure of Transformer encoder.

As shown in Figure 5(a), 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 [21]. Given the input representation of a sequence $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n\}$, Transformer encoder first packs the sequence as a matrix, represented as $\boldsymbol{X} \in \mathbb{R}^{n \times d_m}$, 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 $\boldsymbol{X}$ onto three different subspaces, which can be represented as

$$\{\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}\} = \{\boldsymbol{X}\boldsymbol{W}^Q, \boldsymbol{X}\boldsymbol{W}^K, \boldsymbol{X}\boldsymbol{W}^V\}, \tag{1}$$

where $\boldsymbol{W}^Q$, $\boldsymbol{W}^K$, and $\boldsymbol{W}^V \in \mathbb{R}^{d_m \times d_m}$ are three projection matrices, which project input matrix $\boldsymbol{X}$ onto $\boldsymbol{Q}$, $\boldsymbol{K}$, and $\boldsymbol{V}$, respectively. $\boldsymbol{Q}$, $\boldsymbol{K}$, and $\boldsymbol{V}$ are called *query*, *key*, and *value* as named in Transformer [21]. The output of multi-head self-attention layer can be formulated as

$$\text{MultiHead}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{Concat}\left(\boldsymbol{H}_1, \ldots, \boldsymbol{H}_h\right)\boldsymbol{W}^O, \tag{2}$$

where $\boldsymbol{H}_i, i \in \{1, 2, \ldots, h\}$ is the output of a single scaled dot-product attention head as shown in Figure 5(b) and h is the number of heads. The multi-head self-attention layer concatenates all the outputs $\boldsymbol{H}_i, i \in \{1, 2, \ldots, h\}$ from different heads and then reduces the high dimension feature $\text{Concat}\left(\boldsymbol{H}_1, \ldots, \boldsymbol{H}_h\right)$ to low dimension via another projection matrix $\boldsymbol{W}^O$. To illustrate the dimension of $\boldsymbol{H}_i$ and $\boldsymbol{W}^O$, we first give the formulation of $\boldsymbol{H}_i$ as follows:

$$\begin{aligned} \boldsymbol{H}_i &= \text{Attention}\left(\boldsymbol{Q}\boldsymbol{W}_i^Q, \boldsymbol{K}\boldsymbol{W}_i^K, \boldsymbol{V}\boldsymbol{W}_i^V\right) \\ &= \text{softmax}\left[\frac{\boldsymbol{Q}\boldsymbol{W}_i^Q\left(\boldsymbol{K}\boldsymbol{W}_i^K\right)^\top}{\sqrt{d_k}}\right]\boldsymbol{V}\boldsymbol{W}_i^V. \end{aligned} \tag{3}$$

For each attention head, the original input $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}$ are further projected onto different subspaces via projection matrices $\boldsymbol{W}_i^Q, \boldsymbol{W}_i^K \in \mathbb{R}^{d_m \times d_k}, \boldsymbol{W}_i^V \in \mathbb{R}^{d_m \times d_v}$ so that different heads deal with different input to learn richer information [21]. 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), the output of each attention head $\boldsymbol{H}_i$ is a $d_v \times d_m$ matrix. The concatenated matrix $\text{Concat}\left(\boldsymbol{H}_1, \ldots, \boldsymbol{H}_h\right)$ is a $n \times hd_v$ matrix and $\boldsymbol{W}^O$ is a $hd_v \times d_m$ projection matrix.

As for BERT, the architecture is shown in Figure 5(c), which is based on stacked Transformer encoder blocks [21] and hence incorporates the superiority of multi-head self-attention. In addition,

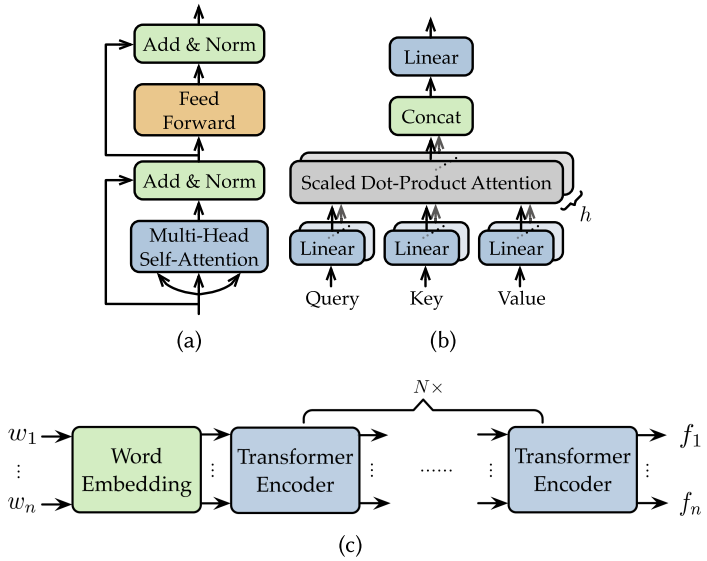


Fig. 5. (a) Transformer Encoder. (b) Multi-Head Self-Attention. (c) BERT.

another significant advantage of BERT [22] 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) [27] 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.

3 DRC-SG FRAMEWORK

To design a powerful key information extractor, both data and architecture design are important. In Section 3.1, we first consider how to label the design rules for effectively training the extractor. Then, in Section 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. In addition to the extractor, in Section 3.4, we explain how to design a rule-based translator to generate the scripts based on the extracted information.

3.1 Semantic Roles

As illustrated in Section 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 [28]. To clearly classify different words, we first clarify all essential semantic roles for rules in FreePDK15 [28].

Some prior works for semantic role labeling studies [29, 30] 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

Table 2. Explanations of All Semantic Roles Defined in Our Work

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.

The bold parts belong to the roles defined in their rows.

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 2.

3.2 Rule Data Generation

Open-source design rules for academic research are relatively rare. Our training dataset, FreePDK15 [28], 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, 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 6(a). 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, then 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.

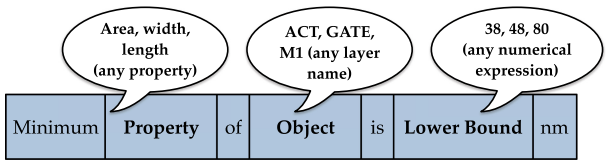
Paraphrasing. In contrast to word order adjustment, paraphrasing can produce a new rule that does not change the meaning but has different expressions. To be specific, we can replace some words with synonyms or change the sentence structure, e.g., from passive to active voice, and an

Original Rule	GATEC shape bottom or top must be aligned if distance is less than 192 nm
Reordered	If distance is less than 192 nm, GATEC shape bottom or top must be aligned

(a)

Original Rule	Minimum vertical width of M1A is 48 nm.
Paraphrasing	The vertical width of M1A is at least 48 nm

(b)



(c)

Fig. 6. Three rule data generation methods. (a) Word Order Adjustment, (b) Paraphrasing, and (c) Template Filling.

example is given in Figure 6(b). To reduce the manpower work, we first rely on a paraphrasing tool called *QuillBot* [31] 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 6(c). 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 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 2,830 new design rules, which effectively expand our rule dataset and contribute to training our key information extractor.

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 7.

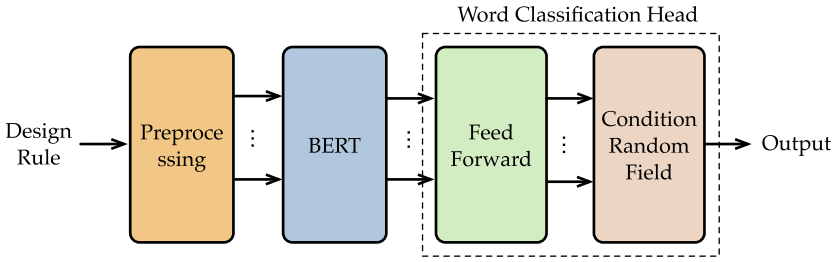


Fig. 7. Architecture of the key information extractor in DRC-SG.

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

$$\text{Preprocess}(\boldsymbol{r}) = [w_1, w_2, \ldots, w_{len(\boldsymbol{r})}, \underbrace{[\text{PAD}], \ldots, [\text{PAD}]}_{L-len(\boldsymbol{r})}], \tag{4}$$

where $\boldsymbol{r}$ is the input design rule. $w_i, i \in \{1, 2, \ldots, len(\boldsymbol{r})\}$ represents each word of $\boldsymbol{r}$, and $len(\boldsymbol{r})$ 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 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 [22], as the feature extractor, which proves to have prominent feature extraction ability according to many natural language processing works like References [25, 26]. As explained in Section 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 $\boldsymbol{F}^o \in \mathbb{R}^{L \times d_b}$, 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 $\boldsymbol{F}^o$ into a word classifier, which is a simple feed-forward neural network composed of two fully connected layers. The output is represented as $\boldsymbol{P}^{wc} \in \mathbb{R}^{L \times N_{wc}}$, where N_{wc} is the number of categories, and the element $\boldsymbol{P}^{wc}_{i,j}$ 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)** [32], on top of the word classifier, whose parameters are a label transition matrix, represented as $\boldsymbol{K} \in \mathbb{R}^{(N_{wc}+2) \times (N_{wc}+2)}$. The element of the label transition matrix $\boldsymbol{K}_{i,j}$

describes the probability of transitioning from label i to j. Two additional states included in $\boldsymbol{K}$ stand for the "start" and "end" of the sequence. In such a case, given a design rule $\boldsymbol{r}$, the probability of a prediction sequence $\boldsymbol{y}$ is calculated from softmax function as

$$p(\boldsymbol{y}|\boldsymbol{r}) = \frac{\exp S(\boldsymbol{r}, \boldsymbol{y})}{\sum_{\hat{\boldsymbol{y}} \in Y_{\boldsymbol{r}}} \exp S(\boldsymbol{r}, \hat{\boldsymbol{y}})}, \tag{5}$$

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

$$S(\boldsymbol{r}, \boldsymbol{y}) = (\boldsymbol{K}_{start, \boldsymbol{y}_1} + \sum_{i=1}^{L-1} \boldsymbol{K}_{\boldsymbol{y}_i, \boldsymbol{y}_{i+1}} + \boldsymbol{K}_{\boldsymbol{y}_L, end}) + \sum_{i=1}^{L} \boldsymbol{P}^{wc}_{i, \boldsymbol{y}_i}. \tag{6}$$

In this way, $S(\boldsymbol{r}, \boldsymbol{y})$ 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 ground-truth probability $p(\boldsymbol{g}|\boldsymbol{r})$, where $\boldsymbol{g}$ is the actual label sequence for the rule $\boldsymbol{r}$. 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:

$$\begin{aligned} \mathcal{L}_{wc} = \mathcal{L}_{crf} &= -\log p(\boldsymbol{g}|\boldsymbol{r}) \\ &= -S(\boldsymbol{r}, \boldsymbol{g}) + \log\left(\sum_{\hat{\boldsymbol{y}} \in Y_{\boldsymbol{r}}} \exp S(\boldsymbol{r}, \hat{\boldsymbol{y}})\right). \end{aligned} \tag{7}$$

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 [33] 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 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 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 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` 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.

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

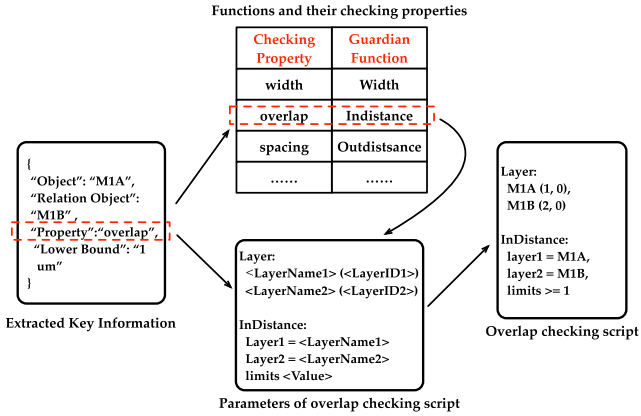


Fig. 8. Script translator.

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.

4.1 Rule Classification Head for Key Information Extractor

In addition to the concrete rule descriptions as shown in the "Examples" column in Table 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 9. It can be seen that the rule classification head shares the same backbone with the word classification head. To predict the rule type, we are supposed to learn the rule feature $\boldsymbol{f}^{rc}$ by combining word features $\boldsymbol{f}_i^o, \forall i \in \{1, 2, \ldots, L\}$ output by the backbone. To achieve this, we rely on the self-attention mechanism illustrated in Section 2.2 and make some customizations.

First, instead of utilizing the entire word features $\boldsymbol{f}_i^o, \forall i \in \{1, 2, \ldots, L\}$, 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 $\boldsymbol{P}^{wc}$ output by the word classification head to guide the word feature selection. $\boldsymbol{P}^{wc}$ 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 informative k words for rule type prediction.

Then, we introduce an extra variable $\boldsymbol{q}^{rc}$ as the *query*, and features $\boldsymbol{f}_i^k, i \in \{1, 2, \ldots, k\}$ from $\boldsymbol{F}^k$ are regarded as the *key* and *value*. Different from the original multi-head self-attention computation paradigm as formulated in Equation (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:

$$\boldsymbol{f}^{rc} = \sum_{i=1}^{k} \frac{\exp(\boldsymbol{q}^{rc}(\boldsymbol{f}_i^k \boldsymbol{W}^K)^\top / \sqrt{d_b})}{\sum_{j=1}^{k} \exp(\boldsymbol{q}^{rc}(\boldsymbol{f}_j^k \boldsymbol{W}^K)^\top / \sqrt{d_b})} \boldsymbol{f}_i^k \boldsymbol{W}^V, \tag{8}$$

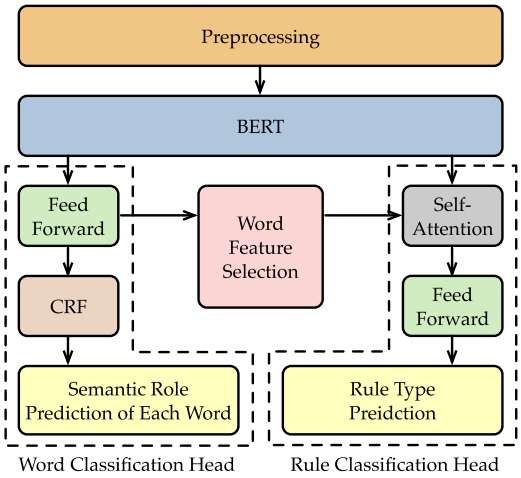


Fig. 9. Architecture of the key information extractor with rule prediction head in DRC-SG 2.0.

ALGORITHM 1: Word Feature Selection Algorithm

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

$\boldsymbol{F}^{k} \leftarrow$ Initialized to empty set;
$\boldsymbol{p}^{None} \leftarrow$ probabilities of all words belonging to "None" class.
$\boldsymbol{SelectIdx} \leftarrow$ indices of the minimum k probabilities in $\boldsymbol{p}^{None}$;
for $i \leftarrow 1, 2, \ldots k$ **do**
    $idx \leftarrow \boldsymbol{SelectIdx}[i]$;
    $\boldsymbol{f}_i^{o} \leftarrow$ the feature in the position idx of $\boldsymbol{F}^{o}$;
    append feature $\boldsymbol{f}_i^{o}$ to $\boldsymbol{F}^{k}$;
end for
return feature set $\boldsymbol{F}^{k}$ with k word features.

where $\boldsymbol{f}^{rc}$ and $\boldsymbol{q}^{rc} \in \mathbb{R}^{d_b}$ have the same dimension as $\boldsymbol{f}_i^{k}$. $\boldsymbol{W}^{K}$ and $\boldsymbol{W}^{V} \in \mathbb{R}^{d_b \times d_b}$ are projection matrices as explained in Section 2.2. Noted that $\boldsymbol{q}^{rc}$ is a learnable variable whose value is determined after extractor training.

Once the representation $\boldsymbol{f}^{rc}$ 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 $\boldsymbol{p}^{rc} \in \mathbb{R}^{N_{rc}}$, 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:

$$\mathcal{L} = \mathcal{L}_{wc} + \lambda \mathcal{L}_{rc}, \tag{9}$$

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

$$\mathcal{L}_{rc} = -\log \frac{\exp\left(\boldsymbol{p}_t^{rc}\right)}{\sum_{i=1}^{N_{rc}} \exp\left(\boldsymbol{p}_i^{rc}\right)}, \tag{10}$$

Table 3. Semantic Roles Distribution of Dataset

Semantic Roles	Training Set		Test Set	
	#	Percentage (%)	#	Percent (%)
Object	6,054	15.68	491	14.74
Relation Object	2,561	6.63	181	5.43
Property	4,705	12.18	300	9.01
Condition	5,061	13.10	596	17.89
Restriction	1,404	3.64	159	4.77
Lower Bound	2,482	6.43	326	9.79
Upper Bound	1,144	2.96	8	0.24
Exact Value	1,430	3.70	40	1.20
None	13,778	35.68	1,230	36.93
Total	38,619	100	3,331	100

where t stands for the ground-truth rule type of the input and $\boldsymbol{p}^{rc}$ is the prediction result of the rule classification head.

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 cannot 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, 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 toward 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:

$$\mathcal{L}_{wc} = \mathcal{L}_{crf} + \eta \mathcal{L}_{imb}, \tag{11}$$

where $\mathcal{L}_{crf}$ is still from Equation (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 (12) and $\boldsymbol{w}_{\boldsymbol{g}_i}$ is the corresponding weight of each term,

$$\mathcal{L}_{imb} = -\sum_{i=1}^{L} \boldsymbol{w}_{\boldsymbol{g}_i} \log \frac{\exp\left(\boldsymbol{P}^{wc}_{i,\boldsymbol{g}_i}\right)}{\sum_{j=1}^{N_{wc}} \exp\left(\boldsymbol{P}^{wc}_{i,j}\right)}, \tag{12}$$

$$\boldsymbol{w}_{\boldsymbol{g}_i} = \exp\left(\frac{1 - \boldsymbol{u}_{\boldsymbol{g}_i} / \max(\boldsymbol{u})}{\alpha}\right). \tag{13}$$

We utilize the ground-truth label of each word to supervise the input of the CRF module, $\boldsymbol{P}^{wc} \in \mathbb{R}^{L \times N_{wc}}$, which includes scores of words belonging to each category. $\boldsymbol{g}_i$ is the ground-truth label of the ith word and $\boldsymbol{u} \in \mathbb{R}^{N_{wc}}$ 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 $\boldsymbol{w}_{\boldsymbol{g}_i}$ 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:

$$\mathcal{L} = \mathcal{L}_{wc} + \lambda \mathcal{L}_{rc} = \mathcal{L}_{crf} + \eta \mathcal{L}_{imb} + \lambda \mathcal{L}_{rc}. \tag{14}$$

The objective of training is to minimize the loss calculated in Equation (14), which can be successfully solved by Adam [34] optimizer, a widely used gradient descent optimization algorithm.

Table 4. Word Classification Comparison Results with Two Other Baselines

Semantic Roles	Bi-RNN [37]			Bi-LSTM [38]			DRC-SG [20]			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**

5 EXPERIMENT RESULTS

5.1 Experimental Settings and Benchmark

We implement our entire framework with the Pytorch library [35] 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 2,970 design rules, 2,840 of which are obtained via our proposed rule generation methods and the rest are the original data from PDK15 [28]. To evaluate the performance, another design kit, ASAP7 [36], acts as the test set, which includes 200 design rules on the 7-nm node. Due to the advanced technology node, rules in ASAP7 are more complex compared with our rules on the 15-nm 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. It can also be observed from Table 3 that the "None" category words accounts for nearly 40%, which further proves that a key information extractor can filter a lot of unnecessary information and contribute to the design rules scripts generation.

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 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.

Word Classification Results. Table 4 summarizes the detailed comparing results in the test set. Since our preliminary work DRC-SG [20] 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 LSTM (Bi-LSTM)**. Similarly to BERT, Bi-RNN [37] and Bi-LSTM [38] 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 remaining two columns, "DRC-SG" and column "DRC-SG 2.0," denote the methods in Reference [20] 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 [20], 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 that contains 200 design rules. The total runtime result is 1.0 s, from which we can calculate that our model averagely takes only 5.0 ms 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 highly efficient. According to our measurement, the script translator spends around 0.46 ms processing one item of key information. In conclusion, by combining the extractor and translator, our framework can generate a single script in 5.46 ms 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 cannot guarantee absolutely accurate results and requires post-correction; however, because of the high accuracy as listed in Table 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.

5.3 Ablation Studies

The major work of this article 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 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 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

Table 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**

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.

6 CONCLUSION

In this article, 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 7-nm technology node have confirmed the excellent performance of our framework: It only takes on average 5.46 ms 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.

REFERENCES

[1] KLayout. Retrieved from https://www.klayout.de/doc/manual/drc.html.
[2] LayoutEditor. Retrieved from https://www.layouteditor.org/layoutscript/api/drc.
[3] Yibo Lin, Shounak Dhar, Wuxi Li, Haoxing Ren, Brucek Khailany, and David Z. Pan. 2019. DREAMPlace: Deep learning toolkit-enabled GPU acceleration for modern VLSI placement. In *Proceedings of the ACM/IEEE Design Automation Conference (DAC'19)*. 1–6.
[4] Siting Liu, Qi Sun, Peiyu Liao, Yibo Lin, and Bei Yu. 2021. Global placement with deep learning-enabled explicit routability optimization. In *Proceedings of the IEEE/ACM Proceedings Design, Automation and Test in Eurpoe (DATE'21)*. 1821–1824.
[5] Zhiyao Xie, Yu-Hung Huang, Guan-Qi Fang, Haoxing Ren, Shao-Yun Fang, Yiran Chen, and Jiang Hu. 2018. RouteNet: Routability prediction for mixed-size designs using convolutional neural network. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD'18)*. 1–8.
[6] Daijoon Hyun, Yuepeng Fan, and Youngsoo Shin. 2019. Accurate wirelength prediction for placement-aware synthesis through machine learning. In *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE'19)*. 324–327.
[7] Yuzhe Ma, Jhih-Rong Gao, Jian Kuang, Jin Miao, and Bei Yu. 2017. A unified framework for simultaneous layout decomposition and mask optimization. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD'17)*. 81–88.
[8] Haoyu Yang, Shuhe Li, Zihao Deng, Yuzhe Ma, Bei Yu, and Evangeline F. Y. Young. 2020. GAN-OPC: Mask optimization with lithography-guided generative adversarial nets. In *Proceedings of the IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD'20)*. 2822–2834.
[9] Guojin Chen, Wanli Chen, Yuzhe Ma, Haoyu Yang, and Bei Yu. 2020. DAMO: Deep agile mask optimization for full chip scale. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD'20)*. 1–9.

[10] Hao Geng, Wei Zhong, Haoyu Yang, Yuzhe Ma, Joydeep Mitra, and Bei Yu. 2020. SRAF insertion via supervised dictionary learning. In *Proceedings of the IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD'20)*. 2849–2859.

[11] Haoyu Yang, Jing Su, Yi Zou, Yuzhe Ma, Bei Yu, and Evangeline F. Y. Young. 2019. Layout hotspot detection with feature tensor generation and deep biased learning. In *Proceedings of the IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD'19)*. 1175–1187.

[12] Hao Geng, Haoyu Yang, Lu Zhang, Jin Miao, Fan Yang, Xuan Zeng, and Bei Yu. 2020. Hotspot detection via attention-based deep layout metric learning. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD'20)*. 1–8.

[13] Ran Chen, Wei Zhong, Haoyu Yang, Hao Geng, Fan Yang, Xuan Zeng, and Bei Yu. 2021. Faster region-based hotspot detection. In *Proceedings of the IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD'21)*. 669–680.

[14] Yiyang Jiang, Fan Yang, Hengliang Zhu, Bei Yu, Dian Zhou, and Xuan Zeng. 2019. Efficient layout hotspot detection via binarized residual neural network. In *Proceedings of the 56th ACM/IEEE Design Automation Conference (DAC'19)*. 1–6.

[15] Christopher B. Harris and Ian G. Harris. 2016. Glast: Learning formal grammars to translate natural language specifications into hardware assertions. In *Proceedings of the IEEE/ACM Proceedings Design, Automation and Test in Eurpoe (DATE'16)*. 966–971.

[16] Junchen Zhao and Ian G. Harris. 2019. Automatic assertion generation from natural language specifications using subtree analysis. In *Proceedings of the IEEE/ACM Proceedings Design, Automation and Test in Eurpoe (DATE'19)*. 598–601.

[17] Rahul Krishnamurthy and Michael S. Hsiao. 2020. Transforming natural language specifications to logical forms for hardware verification. In *Proceedings of the IEEE 38th International Conference on Computer Design (ICCD'20)*. 393–396.

[18] Aysa Fakheri Tabrizi, Logan Rakai, Nima Karimpour Darav, Ismail Bustany, Laleh Behjat, Shuchang Xu, and Andrew Kennings. 2018. A machine learning framework to identify detailed routing short violations from a placed netlist. In *Proceedings of the ACM/IEEE Design Automation Conference (DAC'18)*. 1–6. DOI: http://dx.doi.org/10.1109/DAC.2018.8465835

[19] Riadul Islam and Md Asif Shahjalal. 2019. Late breaking results: Predicting DRC violations using ensemble random forest algorithm. In *Proceedings of the ACM/IEEE Design Automation Conference (DAC'19)*. 1–2.

[20] Binwu Zhu, Xinyun Zhang, Yibo Lin, Bei Yu, and Martin Wong. 2022. Efficient design rule checking script generation via key information extraction. In *Proceedings of the ACM/IEEE 4th Workshop on Machine Learning for CAD (MLCAD'22)*. 77–82. DOI: http://dx.doi.org/10.1109/MLCAD55463.2022.9900085

[21] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NIPS)*. 5998–6008.

[22] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL'19)*. 4171–4186.

[23] Peter J. Liu, Mohammad Saleh, Etienne Pot, Ben Goodrich, Ryan Sepassi, Lukasz Kaiser, and Noam Shazeer. 2018. Generating wikipedia by summarizing long sequences. In *Proceedings of the International Conference on Learning Representations (ICLR'18)*.

[24] Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St John, Noah Constant, Mario Guajardo-Céspedes, Steve Yuan, Chris Tar, et al. 2018. Universal sentence encoder. *arXiv preprint arXiv:1803.11175*.

[25] Wei Yang, Yuqing Xie, Aileen Lin, Xingyu Li, Luchen Tan, Kun Xiong, Ming Li, and Jimmy Lin. 2019. End-to-end open-domain question answering with BERTserini. In *Proceedings of the Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL'19)*.

[26] Jinhua Zhu, Yingce Xia, Lijun Wu, Di He, Tao Qin, Wengang Zhou, Houqiang Li, and Tieyan Liu. 2019. Incorporating BERT into neural machine translation. In *Proceedings of the International Conference on Learning Representations (ICLR'19)*.

[27] Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV'15)*. 19–27.

[28] Kirti Bhanushali. 2014. *Design Rule Development for FreePDK15: An Open Source Predictive Process Design Kit for 15nm FinFET Devices*. Ph.D. Dissertation.

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

[30] Collin F. Baker, Charles J. Fillmore, and John B. Lowe. 1998. The berkeley framenet project. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL'98)*. 86–90.
[31] QuillBot. Retrieved from https://quillbot.com.
[32] John Lafferty, Andrew McCallum, and Fernando C. N. Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the International Conference on Machine Learning (ICML'01)*.
[33] Silvaco. Guardian. Retrieved from https://silvaco.com/wp-content/uploads/product/pdf/guardian_brief.pdf.
[34] Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.
[35] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703*.
[36] Lawrence T. Clark, Vinay Vashishtha, Lucian Shifren, Aditya Gujja, Saurabh Sinha, Brian Cline, Chandarasekaran Ramamurthy, and Greg Yeric. 2016. ASAP7: A 7-nm finFET predictive process design kit. *Microelectr. J.* 53 (2016), 105–115.
[37] Jie Zhou and Wei Xu. 2015. End-to-end learning of semantic role labeling using recurrent neural networks. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL'15)*. 1127–1137.
[38] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of the Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL'18)*. 2227–2237.

Received 9 October 2022; revised 24 February 2023; accepted 20 March 2023