Embedding Hierarchical Signal to Siamese Network for Fast Name Rectification

Yi-An Chen*, Gung-Yu Pan†, Che-Hua Shih†, Yen-Chin Liao*, Chia-Chih Yen†, Hsie-Chia Chang*
Department of Electronics Engineering, Institute of Electronics, National Chiao Tung University *
andychen.ee02@nctu.edu.tw, ycliaoee92g@gmail.com, hcchang@mail.nctu.edu.tw
Synopsys Inc. † {gpan, matar, jackyen}@synopsys.com

***Abstract*—EDA tools are necessary to assist complicated flow of advanced IC design and verification in nowadays industry. After synthesis or simulation, the same signal could be viewed as different hierarchical names, especially for mixed-language designs. This name mismatching problem blocks automation and needs experienced users to rectify manually with domain knowledge. Even rule-based rectification helps the process but still fails when encountering unseen mismatching types. In this paper, hierarchical name rectification is transformed into the similarity search problem where the most similar name becomes the rectified name. However, naive full search in design with string comparison costs unacceptable time. Our proposed framework embeds name strings into vectors for representing distance relation in a latent space using character *n*-gram and locality-sensitive hashing (LSH), and then finds the most similar signal using nearest neighbor search (NNS) and detailed search. Learning similarity using Siamese network provides general name rectification regardless of mismatching types, while string-to-vector embedding for proximity search accelerates the rectification process. Our approach is capable of achieving 93.43% rectification rate with only 0.052s per signal, which outperforms the naive string search with 2.3% higher accuracy and 4,500 times speed-up.**

***Index Terms*—hierarchical name rectification, similarity learning, nearest neighbor search, locality-sensitive hashing, embedding, Siamese network**

I. Introduction

As the IC design and verification flow becomes complicated in nowadays industry, it is vital to ensure automation to maximize productivity using EDA tools. However, tools could modify the design hierarchy or view the same signal into different names in various stages, such as synthesis and simulation. The circumstance is even worse for mixed-language designs or hardware-assisted emulation. This name mismatching problem blocks automation flows and needs name rectification.

This problem is formulated to rectify mismatching names given original design. The goal of rectification system is to efficiently provide correct rectified names. As the mismatching names can be rectified in short time, the automation flows can be continued.

For simple causes of mismatching, there are several workaround for name rectification. Manual rectification needs users background knowledge and experience to find target signal under possible scopes and costs lots of time. Rule-based rectification automates the flows, but still fails when encountering unseen mismatching types. Spell checker stores a database and quickly searches to rectify typo [1]. However, there are limitless possibilities to name signals. Thus, this name rectification problem cannot be solved with a rectification database.

In contrast to correct the name directly, we can build a general rectification system using similarity search. The naive idea is to compare mismatching name with all hierarchical names in design and choose the most similar one as the rectified name. Edit distance [2]–[4] is a method of estimating similarity of two strings by counting the number of edit operations to transform one string into another. The smaller the edit distance means two strings are more similar. However, it is unrealistic to traverse the whole design for every mismatching signal. In addition, string comparisons are time-consuming to calculate the edit distance.

In order to accelerate the rectification process while preserving generality, we propose a two-level similarity framework. It learns similarity relation using Siamese network [5] optimized with triplet loss [6], and it embeds strings into the latent space by locality-sensitive hashing (LSH) [7] for representing their similarity. In the latent space, nearest neighbor search (NNS) [8] is used to find probable candidates. At last, we perform detailed search among the small number of candidates using precise string comparisons. Our approach provides general rectification regardless of mismatching types through similarity learning, while reduces the timing costs by string-to-vector embedding for similarity search.

The remainder of this paper is organized as follows. Section II illustrates the common mismatching types. Section III elaborates our proposed framework. Section IV shows the experimental results. Section V summarizes this paper and explores future works.

II. Background

Mixed-language hardware designs, different EDA tools and emulators cause names mismatching in stage of synthesis, simulation and emulation. Mismatching types increase and evolve continuously day by day. This section illustrates the common mismatching types and their derivation.

In order to name signals with special characters (e.g. ? or &), designers have to follow the naming rule of hardware language. For example, `top.\A123? [0]` and `\top&.leaf1 .\sig&nal1` are types of Verilog's escape name. Some tools especially hardware emulator ignore the white spaces to avoid parsing error but cause name mismatching.

Some hardware description languages such as HSPICE and VHDL are case-insensitive. Different tools could convert all the characters into either upper or lower case. When designers construct a behavior model in SystemVerilog for the corresponding circuit in HSPICE, the signal names could be mismatching. Besides, advanced designs could contain mixed languages, such that some scopes are case-sensitive while others are not. For example, `SV_Top.VH_SCOPE.sv_inst.sig` could be viewed differently among different tools.

In logic synthesis [9] stage, synthesis tool chooses satisfactory gates with time and area constraints. To further optimize design, the tool may break boundary between modules. However, those ungroup instruction may modify the hierarchy of design in the meantime. For example, `SV_Top.Module_A.sub_module.sv_inst.sig` becomes `SV_Top.Module_A_sub_module.sv_inst.sig`.

In SystemVerilog, generate block helps users to define duplicate signals or modules. However, for emulators may share the signals under generate block and need not unfold generate block, which causes the hierarchical names missing generate block. In sight of this mismatching type, we extend for another type: missing scope.

To completely describe a hierarchical name, simulators and emulators split module names with specific delimiters. Such as Verilog and SystemVerilog represents hierarchical names with dot and some emulators may use other delimiters such as /. When design is emulated and need to compare with original Verilog design, changing delimiters causes name mismatching. For example, `Top.SCOPE.inst.sig_A[0]` becomes `Top\SCOPE\inst\sig_A[0]`

For searching signals in design, users have to type hierarchical names. Nevertheless, some characters look similar, such as `1` and `l`, or they are in near positions on keyboard, and typos may happen. Script-based modification for specific keywords may also cause typo. For example, the above mentioned extra escape name is one kind of script-based typo.

For mixed usage of different tools, unseen or double mismatching types emerge in an endless stream. It is a crucial issue to look for a general rectification framework.

III. Proposed Framework

Fig. 1 shows that our proposed framework comprises two phases: similarity learning and similarity search. In similarity learning phase, it extracts features from strings and learns similarity transformation model using automatically-generated training samples. In similarity search phase, it extracts features from names as in learning phase and embeds them into the latent space by the learned model. NNS is applied to search probable candidates in the latent space and detailed search is taken to find the most similar one.

A. Triplet Generation

In the similarity learning phase, our model learns the similarity and dissimilarity between triplet samples [7], [10], [11]. For each triplet, the anchor sample means the mismatching name transformed from the original signal, the positive sample means the original signal in the design, and negative means any other sibling signal. Our framework auto-generates mismatching names for training the similarity model using known mismatching types.

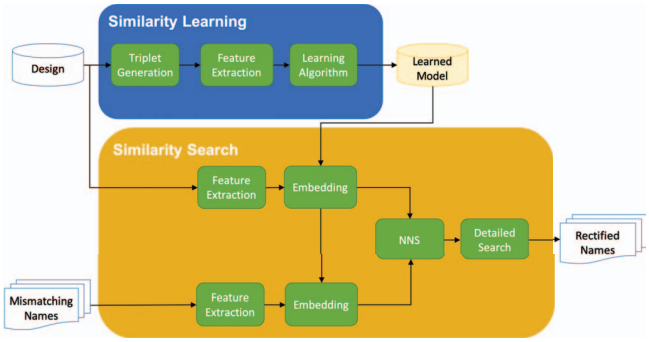

Fig. 1. Our proposed framework consists of two phases: similarity learning and similarity search.

B. Feature Extraction

This subsection details methods of transforming strings into acceptable input format of neural network. In the proposed framework, two feature extraction methods can be chosen: bag-of-characters (BoC) and character *n*-gram.

In text pre-processing, bag-of-words [12] disregards grammar and word order but keeps word multiplicity to nearly represents topic of an article. We benefit from its concept of disregarding order for many IC designers name signal with meaning and even tag with digits to represent different signals. **BoC** method is applied to transform a hierarchical name from a string into a 128-tuple (128 is the number of ASCII). It counts number of each character to fix the length of input and to preserve multiplicity of characters. For example, `array[12]` and `array[21]` are in the same representation with statistical encoding by BoC.

N-gram based method is to neglect order of entire sequence but preserve locally partial features [13]. Its core concept is sliding a window from head to tail of a sequence, where *n* stands for the size of window. Taking `apple` as an example for illustration of **character *n*-gram** [14], character bigram represents it with `ap`, `pp`, `pl`, and `le`.

Considering time complexity, BoC takes *O(n)* and character bigram takes *O(n)*, where *n* is the length of the input string. Next, taking memory complexity into account, BoC takes *128* and character bigram takes 128^2.

C. Learning Algorithm

We apply the concept of **LSH** in similarity learning [7] to learn the transformation of representing the similarity of given samples. Triplet samples are inputted into triplet network whose three networks share their parameters of neural layers as Siamese network. After the triplet network is optimized with triplet loss, its singular network become a transformation network which can embed given input into the latent space.

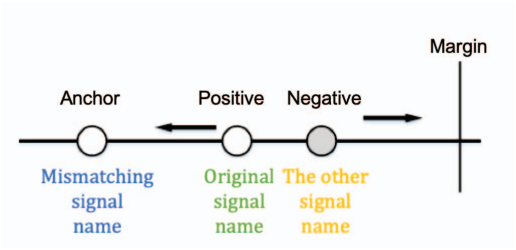

Fig. 2. Triplet loss: Mismatching name is anchor, expected original name is positive sample and any other name is negative sample.

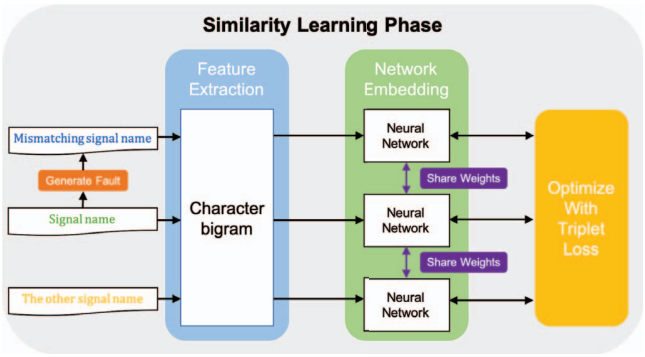

Fig. 3. Triplet Siamese network is used to learn similarity of triplet samples.

Triplet loss [6] is one kind of distance loss function in metric learning [15] defined to optimize model. It calculates loss with implicit distance of two samples in the latent space. Triplet loss helps neural network to become a transformation network for embedding given input into the latent space but also to learn the distance description which is capable of expressing how different of input samples. For one input sample triplet i, the triplet loss L^i is

$$L^i = \left\| f_a^i - f_p^i \right\|_2^2 - \left\| f_a^i - f_n^i \right\|_2^2 + \alpha \tag{1}$$

, where *f* is for vector in space, *a* is for anchor, *p* is for positive sample, *n* is for negative sample and α is for margin between positive pair and negative pair. Its goal is to shorten the distance of pairs expected to be nearer and enlarge the distance of pairs expected to be farther. Fig. 2 illustrates that triplet loss devotes to shorten the distance of mismatching name and original name, and enlarge the distance of mismatching name and any other signal at the same time.

Siamese network [5], [16], [17] is usually used for comparing two hidden features in a latent space to figure out contrastive tasks. It is composed of twin networks which share their parameters of neural layers but accept distinct inputs. The loss function evaluates metric between the features of twin neural networks. In our framework, we use a shallow neural network as triplet Siamese network here for feature extraction is done in the previous stage.

Fig. 3 illustrates the overview of similarity learning phase. After learning the similarity of triplet samples, singular network of triplet network become a transformation network which can embed given input into the latent space.

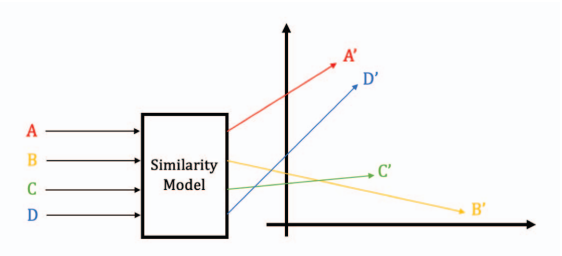

Fig. 4. LSH represents the similarity of inputs with distance in a latent space.

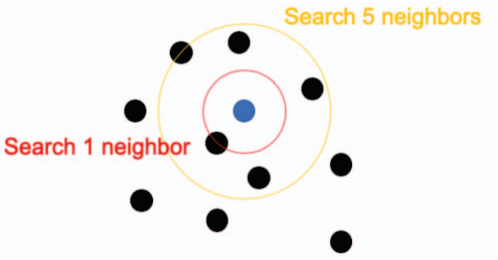

Fig. 5. NNS is used to find more candidates.

D. Embedding

Fig. 4 shows the concept of embedding using LSH. The transformation network encodes the same signal viewed in different names to near positions in the latent space, such as *A* and *D* in Fig. 4. Similarity of hierarchical names is described with distance relation in the latent space. The nearer is the more similar, the farther is the more different.

E. Nearest Neighbor Search

NNS [8] takes low-cost time to find the closest samples in the embedding space. In this paper, we build the k-dimensional tree (k-d tree) [18] to represent the design hierarchy, and search each mismatching signal on this tree for the most similar candidates. For each design, the k-d tree must be built once, and each mismatching name is searched using the tree. Compared to naive full search in design, NNS significantly reduces searching time. Fig. 5 shows the concept of NNS finds more nearest neighbors. Searching for more nearest neighbors provides detailed search with more comparing candidates.

F. Detailed Search

After proximity search in the latent space, we propose **hierarchical reward function** to precisely choose the most similar name among candidates from NNS. It comprises character-based and scope-based similarity. To normalize every part of reward, they are individually divided by their metric of input.

Edit distance [2]–[4] performs well on some slight mismatching types such as extra escape name and ungroup scope. We preserve it as one part of reward. The definition of reward of edit distance R_{ED} is

$$R_{ED} = \frac{l(S_1) - d_e(S_1, S_2)}{l(S_1)} \tag{2}$$

TABLE I
BENCHMARKS

	DFT	2-dimension Convolution	Elliptic Curve Multiplication
Total Signal	60,053	2359	46,204
Signal with genblk	21,005	0	0
Training Data	60,053	2359	46,204
Testing Data	381,323	14,154	277,224

, where l is for the length of string, d_e is for edit distance of two strings, S_1 is for input name string and S_2 is for comparative string.

Also, considering missing scope, we split hierarchical names with delimiters into multiple scopes and count number of identical scope. The definition of reward of matching scope R_{MS} is

$$R_{MS} = \sum_{s_1 \in S_1, s_2 \in S_2} s_1 = s_2 \quad (3)$$

, where s_1 is scope of S_1 and s_2 is scope of S_2.

At last, so as to give sufficient consideration to estimate similarity of two strings, we not only directly compute reward of two strings, but also compute the above two kinds of reward with case-insensitive strings. Lastly, our hierarchical reward function is average of the above parts of reward.

IV. Experimental Results

A. Environmental Settings

Our computing device are CPU: Intel Xeon E5-2620V4 2,1GHz, GPU: GTX 1080 TI and DRAM: DDR4 64GB. In learning phase, both of CPU and GPU are used to compute. In similarity search phase, only CPU is used.

The performance is evaluated with three SystemVerilog designs. One of them is private design and the other two are from: discrete Fourier transform (DFT) intellectual property (IP) generator [19] and elliptic curve multiplication [20]. We discuss and compare the results with the DFT design and finally evaluate performance of the others to the generality. Table I shows the message of the designs. To separate training and evaluation data, we take one type as training data and the others as evaluation data for each signal. There is not existing generate block on each hierarchical name, so we do not use them in learning phase but as evaluation data. In this problem, rectification rate is the correctly rectified number divided by the total number of mismatching names. Top 5 accuracy means that the expected answer must be among 5 highest probable answers.

B. Performance of Similarity Search Methods

Table II shows the rectification rate and processing time of different methods to rectify a mismatching name. NNS provides much faster rectification time with acceptable amortized training time and detailed search provides much preciser string comparison with acceptable time. By NNS in the embedding space and detailed search with hierarchical reward, our framework outperforms naive full search with 2.3% higher rectification rate and 4,500 times speed-up. Our proposed method has an advantage in time as long as the number of mismatching names more than 14.

TABLE II
PERFORMANCE OF DETAILED SEARCH WITH EDIT DISTANCE AND HIERARCHICAL REWARD

	Full search with edit distance	CB (16-dim.) top 1	CB (16-dim.) search among top 5 with edit distance	CB (16-dim.) search among top 5 with hierarchical reward
Training time (ms)	N/A	3,128.85	3,128.85	3,128.85
Rectification time (ms)	$2.36*10^5$	0.22	16.14	24.78
Rectification rate (%)	91.13 (729/800)	74.13	90.78	93.43

TABLE III
PERFORMANCE OF DIFFERENT HYPER-PARAMETERS

	Full search with edit distance	BoC (4-dim.)	BoC (16-dim.)	CB (4-dim.)	CB (16-dim.)
Training time (s)	N/A	2,421.66	2,415.22	3,064.81	3,128.85
Embedding time (ms)	N/A	0.23	0.23	0.32	0.32
Time of search top 1 (ms)	N/A	0.10	0.26	0.091	0.22
Time of search top 5 (ms)	N/A	0.11	0.59	0.10	0.77
Top 1 rectification time (ms)	$2.36*10^5$	0.33	0.49	0.41	0.54
Top 5 rectification time (ms)	N/A	0.34	0.82	0.42	0.99
Top 1 rectification rate (%)	91.13 (729/800)	29.50	35.19	74.03	74.13
Top 5 rectification rate (%)	N/A	56.88	67.15	82.43	93.73

Table III shows the average rectification rate and time of different methods to rectify a mismatching name. The leftmost column is full search in design with edit distance, and the other columns are our works which directly search the nearest samples in the embedding space. Our method is benefited from NNS to take much less processing time than full search with edit distance. Taking the rightmost column (feature extraction by character bigram, embedding to 16-dimension and search the nearest sample as rectified name) to compare, our method processes faster than full search with edit distance 437,000 times. Although the top 1 rectification rate of our method is lower than full search in design with 17%, our framework benefits from distance relation in the embedding space to search for more nearest neighbors. The top 5 rectification rate is with 2.6% higher, and detailed search is applied to choose the most similar name with preciser string comparison.

C. Comparisons on Feature Extraction Methods

N-gram is set 2 (character bigram) here instead of other numbers for actually n=1 is BoC and $n > 2$ costs lots of memory. It is suitable to use character bigram for feature extraction in this problem. Fig. 6 shows the results of two different feature extraction methodologies for each mismatching types. Embedding to 16-dimension latent space, character bigram (CB) significantly outperforms than BoC. The former arrives 93.73% top 5 rectification rate in average; the latter arrives 67.15% top 5 rectification rate in average. CB takes

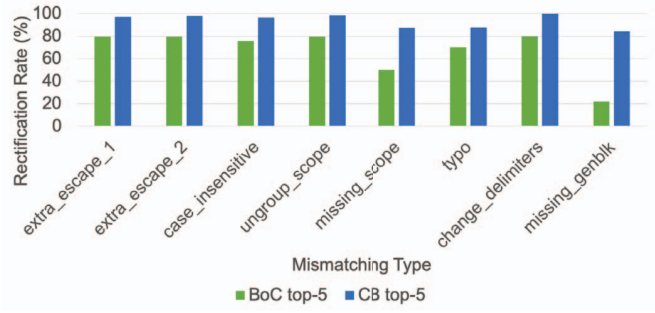

Fig. 6. The result is to search 5 nearest neighbors in 16-dimension embedding space. It shows that feature extraction by character bigram outperforms by BoC for each mismatching type.

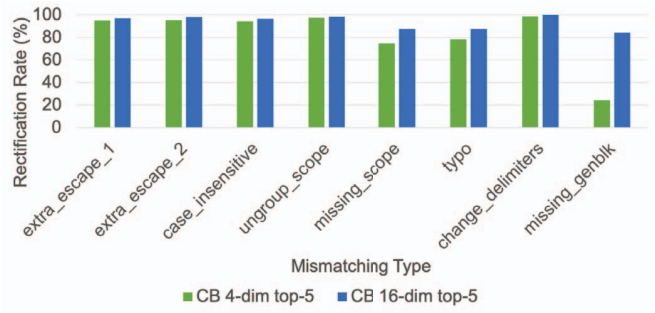

Fig. 7. The rectification rate is benefited from embedding into higher dimension, especially for missing scope and missing genblk.

only 0.17 ms more time to elevate 26.58% top 5 rectification rate.

D. Comparisons on Embedding Dimensions

Missing scope and missing genblk are two most difficult types, and our framework performs well by embedding into higher-dimensional latent space. Fig. 7 shows that their top 5 rectification rate is all raised obviously and over 80% after embedding to 16-dimension. For missing genblk type, its rectification rate can be elevated from 24.38% to 84.33% with only more 0.67 ms to NNS in higer dimension. The trend of rectification rate of missing genblk is not as the others for two reasons: one is their hierarchical names are much longer and similar to each of them, and one is that they are not trained with the transformation network.

E. Analysis on Searching More Samples

The results of detailed search show that our method benefits from searching more candidates in the embedding space and then compares strings with hierarchical reward. Our framework tries to search more nearest neighbors and then applies detailed search. Fig. 8 shows that the overall rectification rate increases by searching more candidates in the embedding space. The bar chart mapping to left-vertical axis represents the rectification rate of top *n*. The line chart mapping to right-vertical axis represents the processing time for name rectification. Applying NNS to search the nearest one provides

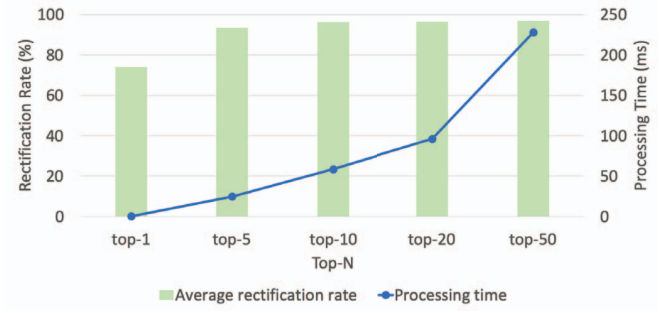

Fig. 8. Detailed searching more nearest neighbors elevates rectification rate but also processing time.

TABLE IV
PERFORMANCE OF FULL SEARCH IN DESIGN WITH HIERARCHICAL REWARD

	Accuracy	Average Time per Signal of One Mismatching Name
Extra Escape Type 1	100%	371.93s
Extra Escape Type 2	100%	368.74s
Case Insensitive	100%	367.63s
Ungroup Scope	100%	370.54s
Missing scope	92%	271.55s
Typo	100%	397.01s
Change Delimiters	100%	382.61s
Missing genblk	90%	408.91s
Average	97.75%(792/800)	370.32s

74.12% rectification rate with 0.22 ms per signal. Implementing detailed search with hierarchical reward among 5 candidates from NNS elevates rectification rate to 93.43% with 24.78 ms. The latter takes only more 24.56 ms to magnificently raise 19.31 % rectification rate. To further search more candidates in the latent space, detailed search among 50 candidates arrives 97.00% rectification rate with 228.00 ms per signal. It is not worth to take 203.22 ms to elevate only 3.57%.

F. Analysis on Detailed Search

We propose hierarchical reward in section III-F to choose the most similar name. It takes a little more time than edit distance to more precisely estimate the similarity of two hierarchical names. Fig. 9 shows that detailed search with hierarchical reward performs better than with edit distance on missing scope and missing genblk. The performance is shown in table II. For top 1 rectification rate, hierarchical reward is better to choose the most similar one. It is worth to cost acceptable time to use hierarchical reward.

Table IV shows the results of full search in design with hierarchical reward function. The average rectification rate is 97.75% and 370.32s per signal. In other words, hierarchical reward function takes 0.0062s to compare two strings. For missing scope type with shorter string, it costs less processing time. Hierarchical reward function improves the drawback of being hard to recognize missing scope type. Even though it takes more time than edit distance, we still can exploit its better similarity estimation between hierarchical names.

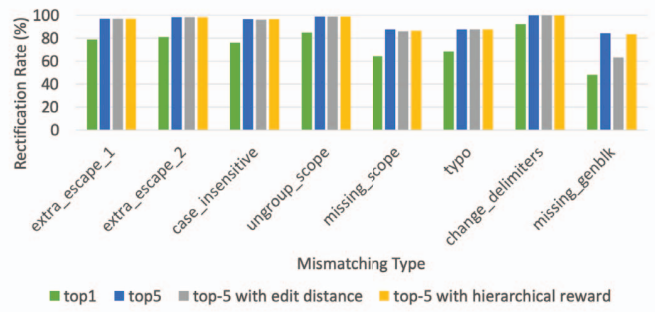

Fig. 9. Detailed search with hierarchical reward outperforms with edit distance on types of missing scope and missing genblk.

TABLE V
PERFORMANCE OF DIFFERENT DESIGNS

	DFT	2-dimension Convolution	Elliptic Curve Multiplication
Training Time (s)	3,128.85	122.03	2,299.25
Rectification Time (ms)	24.78	8.62	11.30
Rectification Rate (%)	93.43	93.95*	95.62*

G. Analysis on Different Designs

Table V (* stands for w/o genblk) shows the rectification results of different designs. Training and evaluation time are proportional to the number of signals in design. For these three different scale designs, our framework correctly rectifies over 93% mismatching names.

V. CONCLUSIONS

In this paper, we model the name rectification problem into similarity search and boost the performance by similarity learning techniques. Our framework encodes strings to vectors to perform fast NNS, where the rectification time is accelerated by vector-based comparisons and proximity search. We exploit n-gram to extract string features where naming characteristics are preserved, and take advantage of LSH to reduce the dimension of vectors. Meanwhile, the similarity is learned by triplet Siamese network with auto-generated training samples. At last, we propose hierarchical reward to perform string-based detailed search among top candidates in NNS. Different hyper-parameters are examined and analyzed to validate the stability of our method and demonstrate trade-off between accuracy and speed. Our framework is capable of achieving 93.43% rectification rate with only 0.052s per signal including training time, which outperforms the naive string search with 2.3% higher rectification rate and 4,500 times speed-up.

The proposed framework can be integrated with existing design and verification flow to facilitate automation. Besides, it is also applicable to interactive tools as the recommendation subsystem; users could choose from top candidates when the target name strings could not be found directly. In real industry, designs are mostly inherited from last generation and enhanced progressively; hence, transfer learning techniques could be applied to further accelerate training time by updating the neural network gradually. In addition, we can report the common mismatching types and even synthesis the rectification rules for users to tune their flow. For advanced applications, the building blocks in our framework can be replaced; for example, recurrent neural network (RNN) could be plugged-in Siamese architecture to capture global string features.

ACKNOWLEDGMENT

This work was supported by the Higher Education Sprout Project of the National Chiao Tung University and Ministry of Education (MOE), Taiwan. Under grant MOST, Taiwan 107-2622-8-009-014-TA.

REFERENCES

[1] I. Prilepov, T.-L. Cheung, K.-F. Lee, S.-L. Tam, and M.-H. Chan, "Web-based spell checker," U.S. Patent, 8 Nov., 2016.
[2] R. A. Wagner and M. J. Fischer, "The string to string correction problem," pp. 168–173, 1974.
[3] W. J. Masek and M. S. Paterson, "A faster algorithm computing string edit distances," *Journal of Computer and System Sciences*, vol. 20, pp. 18–31, Feb. 1980.
[4] D. Chakraborty, D. Das, E. Goldenberg, M. Koucký, and M. E. Saks, "Approximating edit distance within constant factor in truly sub-quadratic time," *Computing Research Repository*, 2018.
[5] J. Bromley, I. Guyon, Y. LeCun, E. Säckinger, and R. Shah, "Signature verification using a "siamese" time delay neural network," in *Advances in Neural Information Processing Systems 6*, 1994, pp. 737–744.
[6] F. Schroff, D. Kalenichenko, and J. Philbin, "FaceNet: A unified embedding for face recognition and clustering," *Computing Research Repository*, 2015.
[7] A. Mignon and F. Jurie, "PCCA: A new approach for distance learning from sparse pairwise constraints," in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, June 2012, pp. 2666–2672.
[8] P. N. Yianilos, "Data structures and algorithms for nearest neighbor search in general metric spaces," in *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, 1993.
[9] P. Kurup and T. Abbasi, *Logic synthesis using Synopsys*, 2nd ed. Springer Publishing Company, Incorporated, 2011.
[10] G. Chechik, V. Sharma, U. Shalit, and S. Bengio, "Large scale online learning of image similarity through ranking," *Journal of Machine Learning Research*, vol. 11, pp. 1109–1135, Mar. 2010.
[11] M. Guillaumin, J. Verbeek, and C. Schmid, "Is that you? metric learning approaches for face identification," *Proceedings of the IEEE International Conference on Computer Vision*, pp. 498–505, Nov. 2009.
[12] Y. Zhang, R. Jin, and Z. cheng Zhou, "Understanding bag-of-words model: a statistical framework," *International Journal of Machine Learning and Cybernetics*, vol. 1, pp. 43–52, Dec. 2010.
[13] W. B. Cavnar and J. M. Trenkle, "N-gram-based text categorization," *In Proceedings of 3rd Annual Symposium on Document Analysis and Information Retrieval*, pp. 161–175, 1994.
[14] M. Popović, "chrF: character n-gram f-score for automatic MT evaluation," in *Proceedings of the Tenth Workshop on Statistical Machine Translation*, Sep. 2015.
[15] E. Hoffer and N. Ailon, "Deep metric learning using triplet network," in *International Conference on Learning Representations*, 2014.
[16] D. Held, S. Thrun, and S. Savarese, "Learning to track at 100 FPS with deep regression networks," *Computing Research Repository*, Apr. 2016.
[17] Y. Wen, K. Zhang, Z. Li, and Y. Qiao, "A discriminative feature learning approach for deep face recognition," in *European Conference on Computer Vision*, 2016.
[18] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Communications of the ACM*, vol. 18, pp. 509–517, Sep. 1975.
[19] Spiral discrete Fourier transform / fast Fourier transform ip generator. [Online]. Available: https://www.spiral.net/hardware/dftgen.html
[20] P. Choi, M. Lee, J. Kim, and D. K. Kim, "Low-complexity elliptic curve cryptography processor based on configurable partial modular reduction over nist prime fields," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 65, no. 11, pp. 1703–1707, Nov. 2018.