# Toward A Reinforcement Learning-based Rectilinear Macro Placement under Human Constraints

Tuyen P. Le, Hieu T. Nguyen, Seungyeol Baek, Taeyoun Kim, Jungwoo Lee (AgileSoDA Company)

Seongjung Kim, Hyunjin Kim, Misu Jung, Daehoon Kim, Seokyong Lee (Asicland Company)

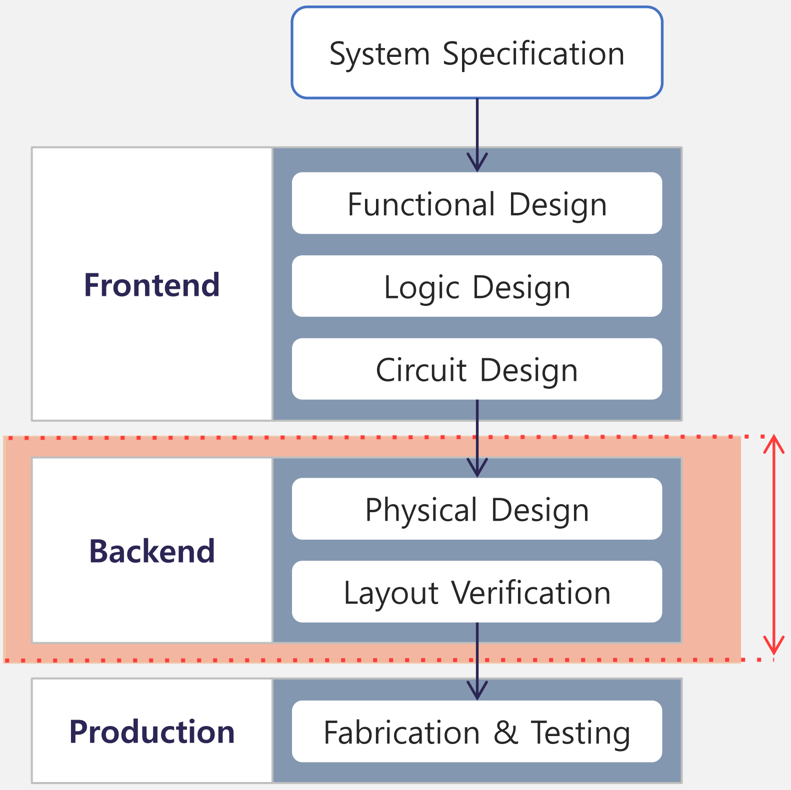
Daewoo Choi (Hankuk University of Foreign Studies)

Agile SoDA

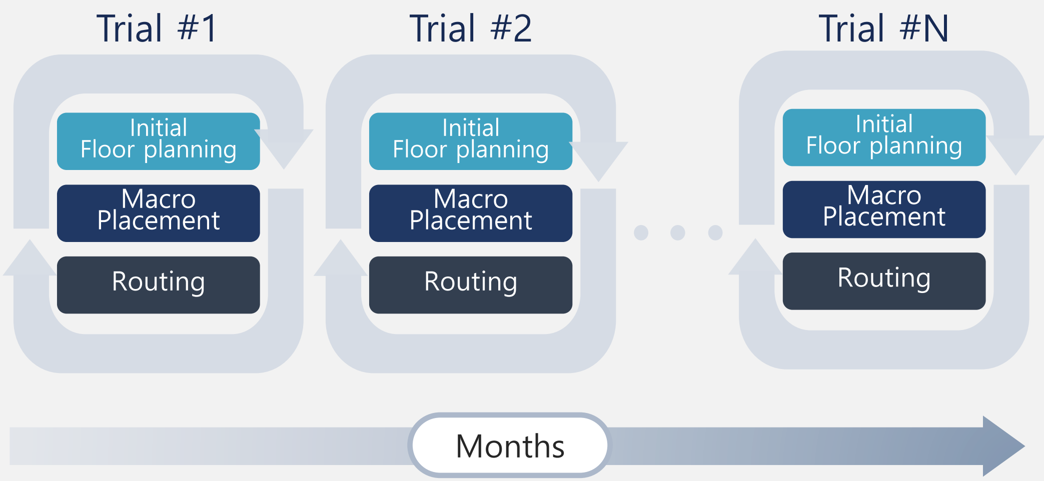
# Outline

# Macro placement is a critical phase in chip design

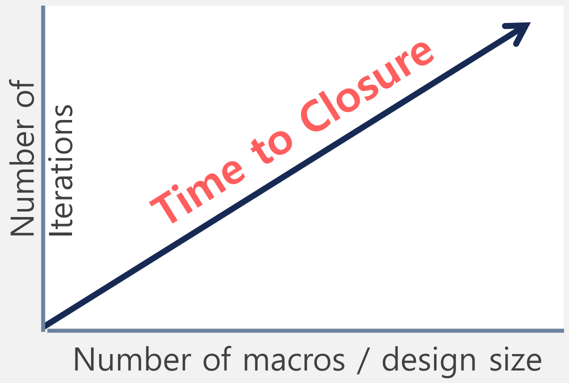
## Semiconductor Design Process




## Time consuming task




- Human designers repeat the placement to get optimal result



# Google's Nature Paper: Deep Reinforcement Learning-based Macro Placement

This method is a chip placement approach that has the ability to generalize, meaning that it can leverage what it has learned while placing previous netlists to generate better placements for new unseen netlists.

**nature**

Explore content | About the journal | Publish with us

nature > articles > article

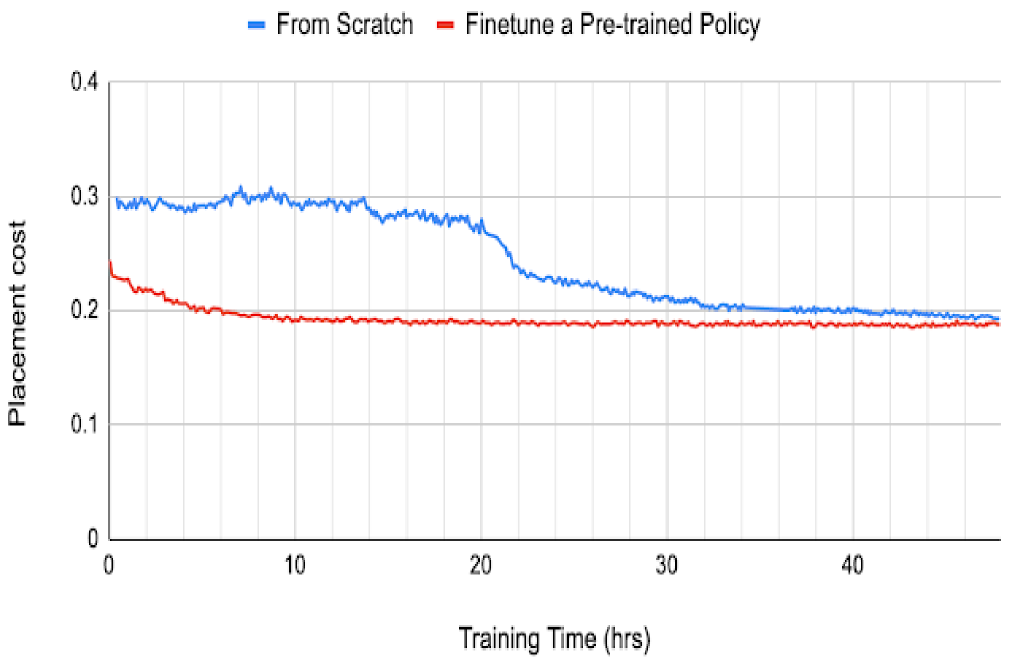
Article | Published: 09 June 2021

## A graph placement methodology for fast chip design

Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Wenjie Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Azade Nazi, Jiwoo Pak, Andy Tong, Kavya Srinivasa, William Hang, Emre Tuncer, Quoc V. Le, James Laudon, Richard Ho, Roger Carpenter & Jeff Dean

*Nature* **594**, 207–212 (2021) | Cite this article

**43k** Accesses | **98** Citations | **2077** Altmetric | Metrics

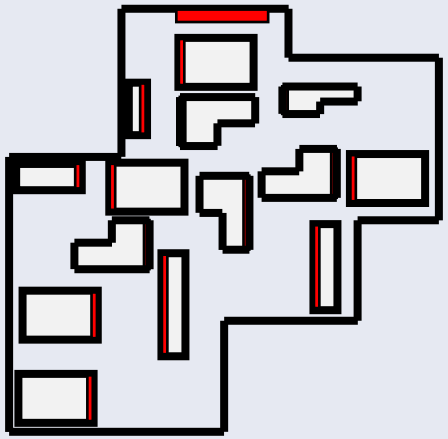





Agile SoDA

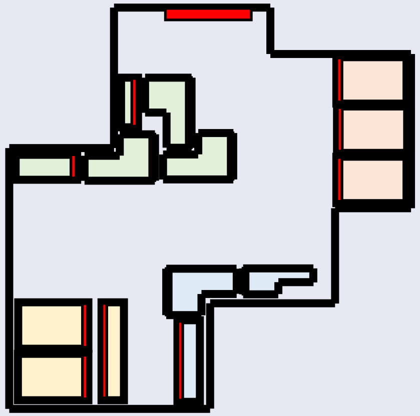
# Motivation

## Dealing with rectilinear macros and layout areas.



✓ Macro placement becomes more intricate when involving general **rectilinear macros and layout areas**

## Human-like constraints



✓ Macro placement that incorporates human-like constraints, such as **design hierarchy** and **peripheral bias**, has the potential to significantly reduce the amount of additional manual labor required from designers.

## Reduce training resources

Training resources from Google Circuit Training

For the training we utilized the following servers and jobs:

- 1 Replay Buffer(Reverb)/Eval server 32vCPUs (n1-standard-32)
  - 1 Replay Buffer(Reverb) job
  - 1 Eval job
- 20 Collect servers 96vCPUs (n1-standard-96)
  - Each server running 25 collect jobs for a total of 500.
- 1 Training server: 8xV100s (n1-standard-96)
  - 1 Training job

✓ We want to constrain training resource utilization to typical configurations

- 01 x A5000 GPU (24GB)
- 01 x 64vCPUs

# Our Efforts

## Enhancements on Google's CT

- We propose enhancements to CT-based macro placement including fine-tuning placement to account for human-like constraints
  - Placing macros based on design hierarchy
  - Placing macros at the periphery

## Rectilinear macros and layout areas

- We present methods to unify macro placement using macros and layout areas for general rectilinear shapes
- To the best of our knowledge, this is the first work dealing with rectilinear layout areas and macro shapes using RL.

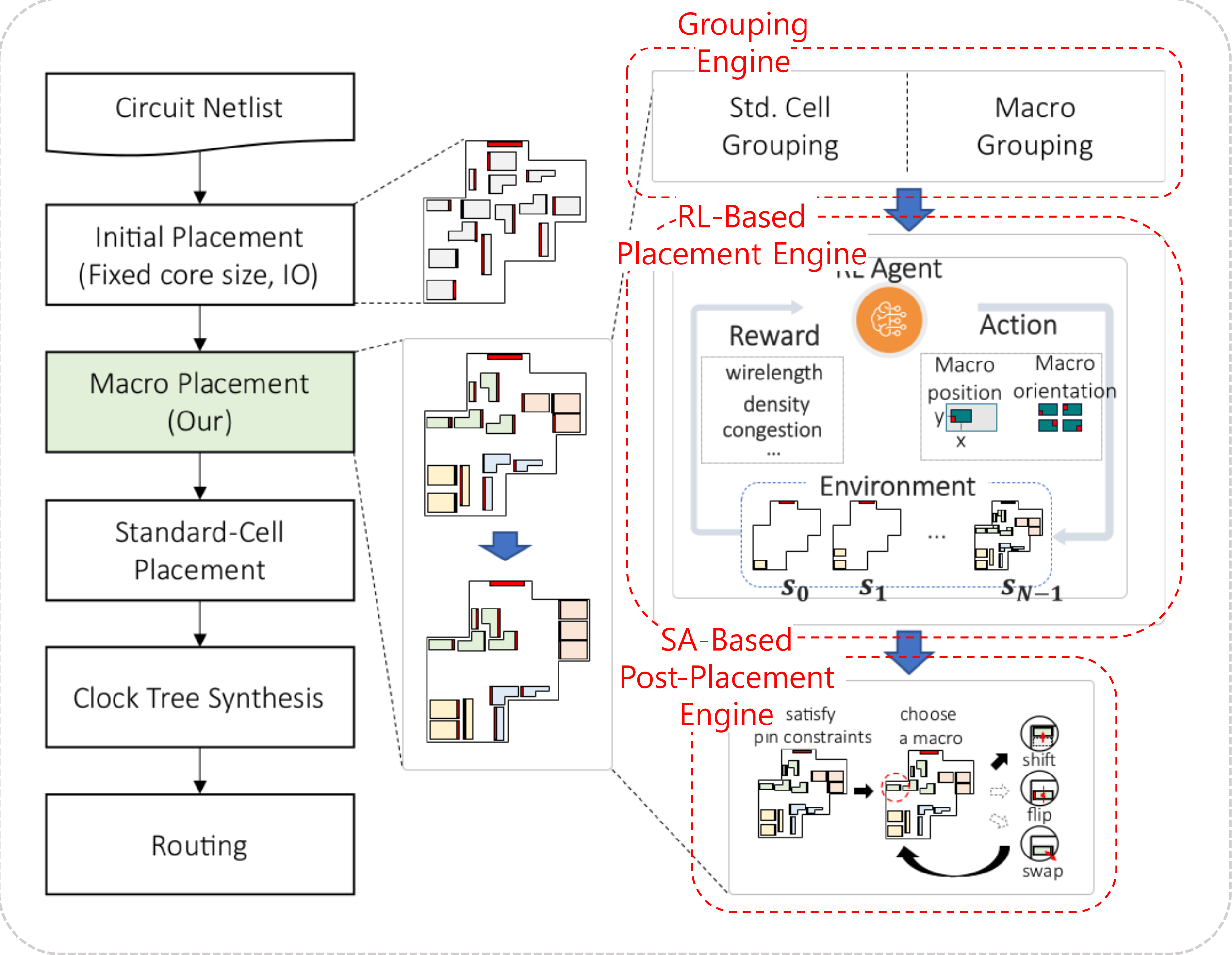
## RL model Enhancement

- We propose an enhanced RL model and demonstrate that our RL-based placer can use fewer resources
- RL model still achieves competitive PPA metrics


# Our Methodology

# Methodology

- Our framework consists of three distinct engines designed to optimize the processes of standard cell and macro grouping, macro placement, and post-processing placement:
  - The grouping engine groups millions of standard cells into several clusters and classifies all the macros into groups based on the design hierarchy
  - The RL-based placement engine receives input from the grouping engine and produces near-final placements. This engine uses methods to handle rectilinear macros and layout areas, and to satisfy constraints about the design hierarchy, and peripheral bias
  - The SA-based post-placement engine fine tunes the results generated by the RL placement engine for better pin accessibility, and dead-space minimization.

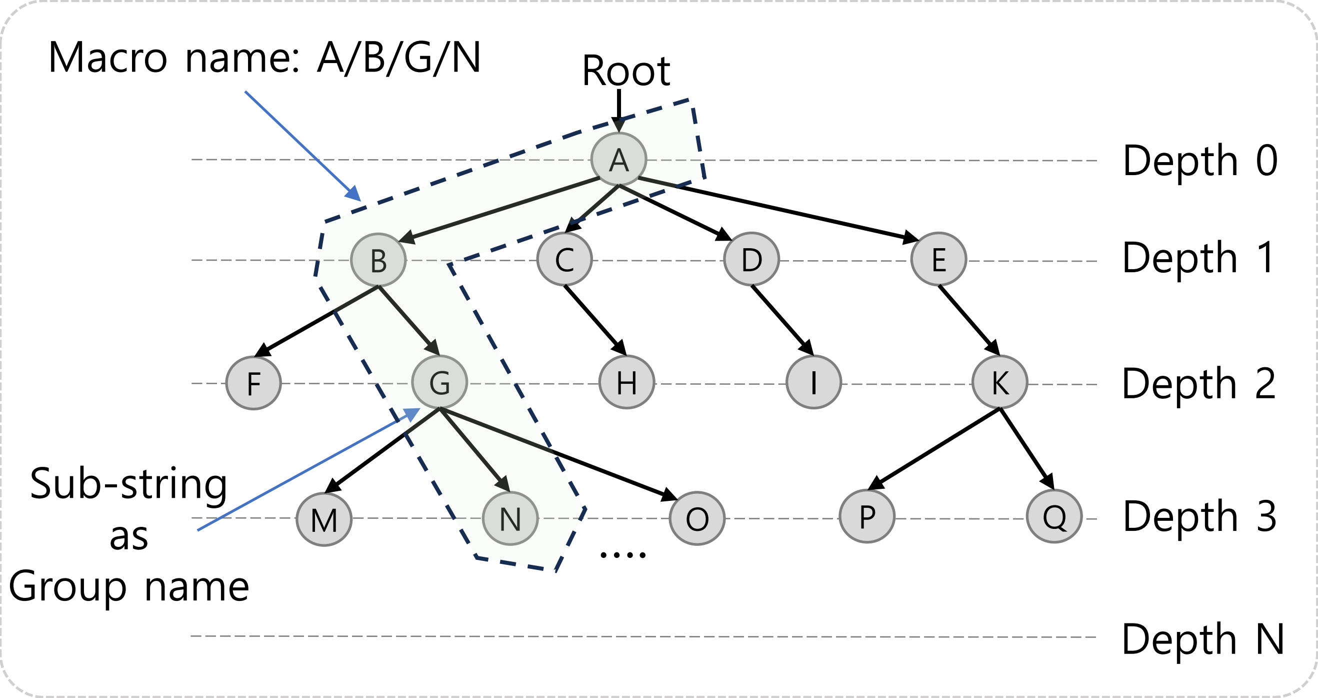





Agile SoDA

# Grouping Macros

Grouping macros can be guided by human or automatically inferred from the netlist (as we did not have access to the original RTL).

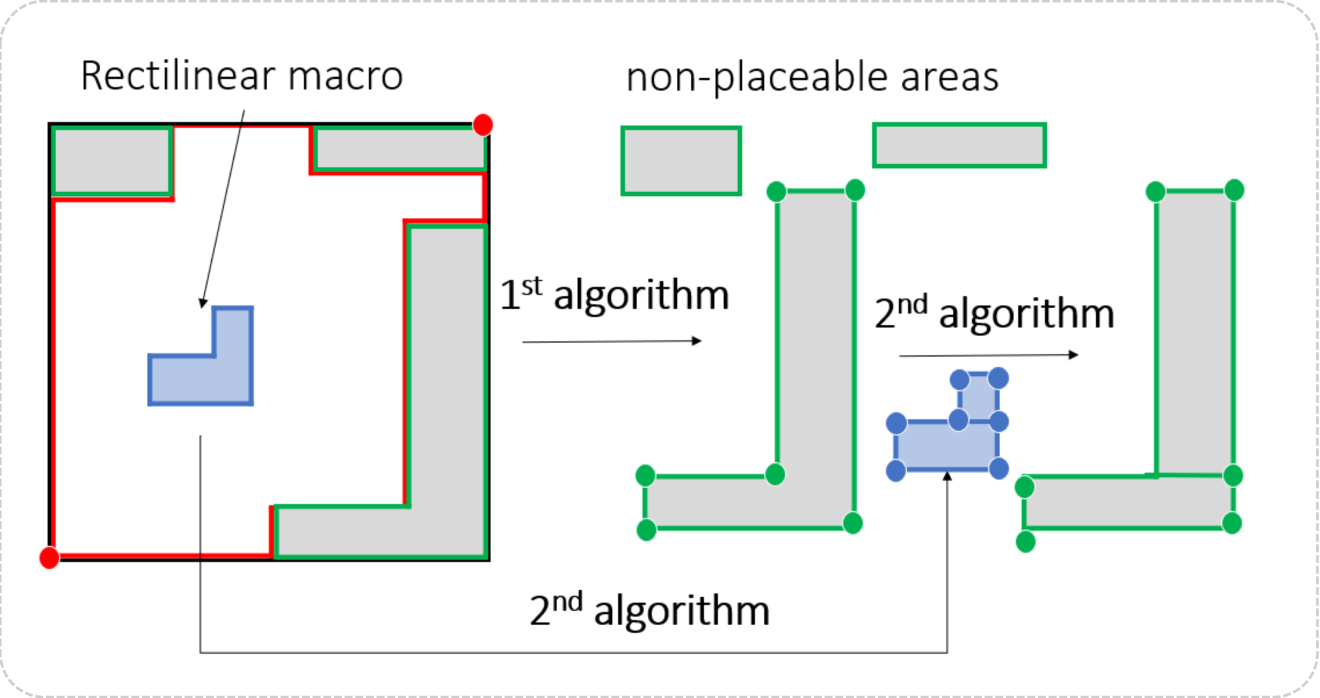



An example of tree data structure of macro names

- Grouping macros based on human when human guidance is possible.
- When human guidance is not possible, we propose an alternative method which analyzes the names of all macros in the netlist
    - Recursive search procedure is implemented at each depth level of the tree.
    - If a node at a given depth level has more than one child, it is considered a group
    - Otherwise, the search continues to deeper depth levels

# Rectilinear Macros and Layout Handling

We propose two algorithms for handling the placement of rectilinear macros, allowing the use of a grid-based masking algorithm (next slide) to work with "primitive", i.e. rectangular, blocks and maximize the use of the layout area.

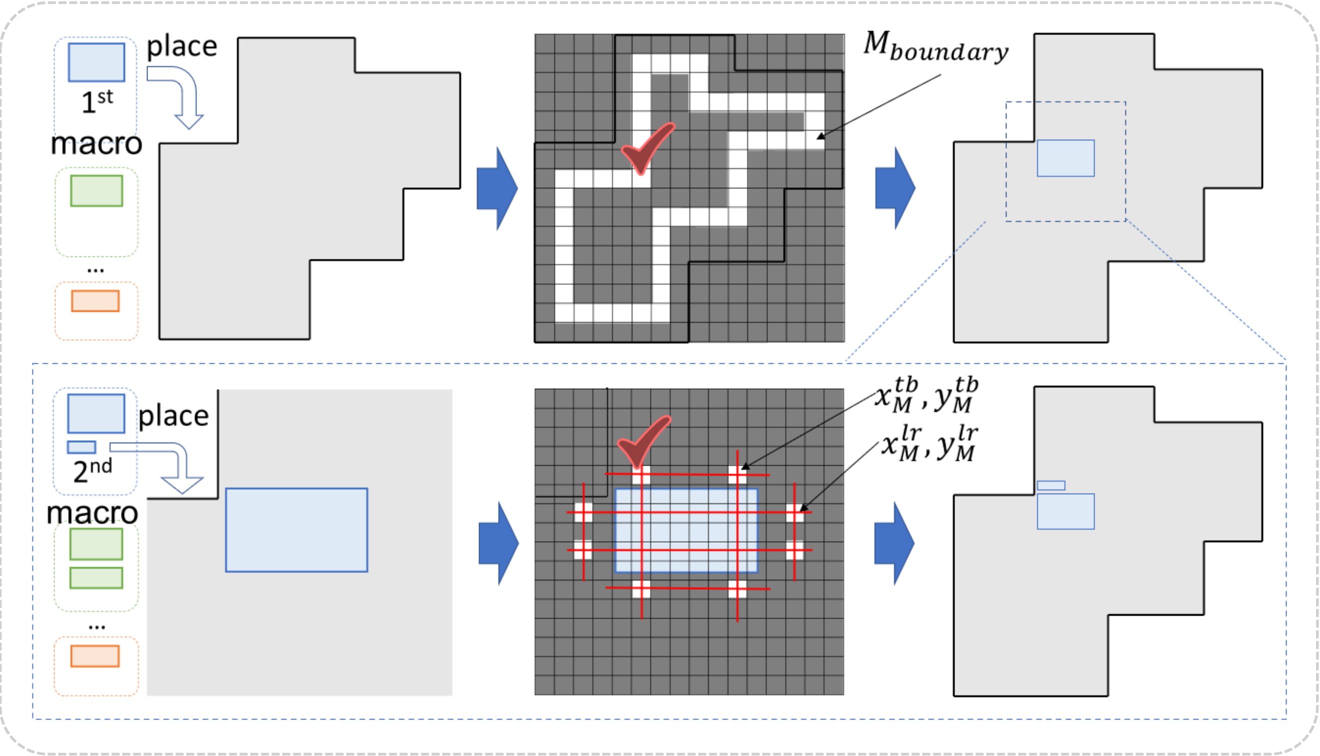



Rectilinear macros and layout handling

- The first algorithm identifies non-placeable areas
- The second algorithm decomposes each rectilinear shape (non-placeable areas and rectilinear macros) into multiple rectangles

# Masking Control Algorithm

We control the position mask to ensure that the currently placed macro adheres to the design hierarchy and periphery bias




An illustration of the masking control algorithm.

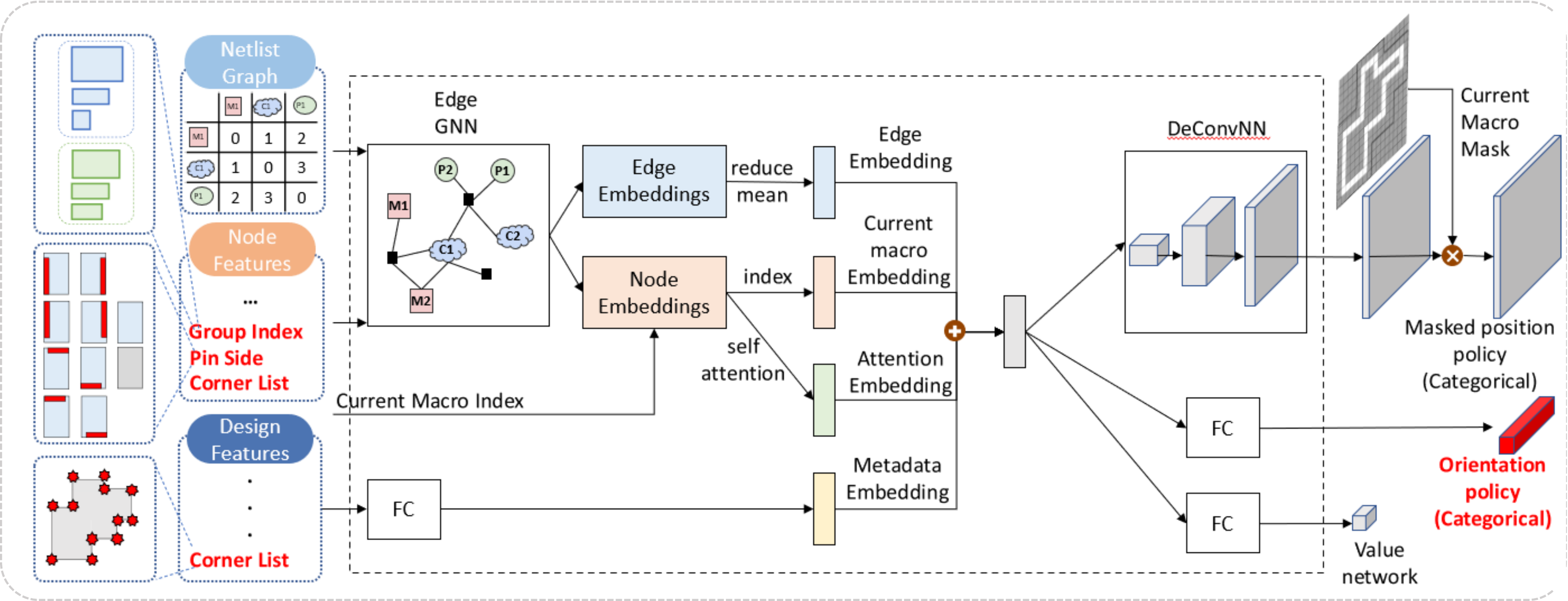
- If the macro is the first from its group:
  - the position mask is the boundary mask ($M_{boundary}$), which allows the macro to be placed only by the closest peripheral grid cells.
- Beginning with the second macro of a group:
  - The algorithm restricts the placeable grid cells to be in close proximity to macros from the same group that have already been placed



Agile SoDA

# Neural network model

Our proposal incorporates additional information that significantly enriches the macro and design features. Furthermore, as an additional advancement, we upgrade our model to a two-head policy.




- Additional information:
  - group index
  - pin side
  - corner list

- Two-head policy:
  - macro position
  - macro orientation



Agile SoDA

# Reward function

Our reward function R is defined as a negative weighted sum of four proxy costs:

$$\mathcal{R} = -(\alpha C_W + \beta C_C + \gamma C_D + \omega C_H)$$

- $C_W$: the wirelength cost is approximated as the normalized half-perimeter wirelength (HPWL)
- $C_D$ : the density cost is approximated as the average density of the densest 10% of grid cells
- $C_C$ : the congestion cost is approximated as the average of the top 5% most congested grid cells
- $C_H$ : the hierarchy cost is to encourage closeness between macros in the same design hierarchy

Number of macro in group $g$

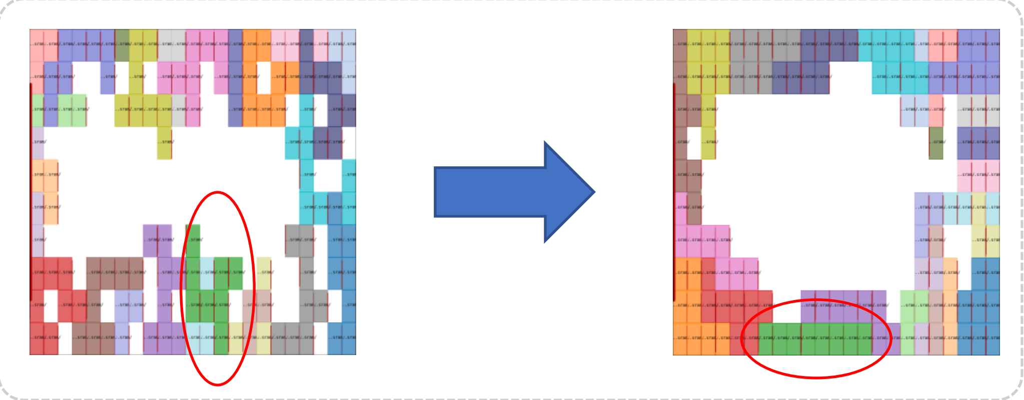
Euclidean distance

$$C_H = \frac{1}{G} \sum_{g=0}^{G} \frac{\sum_{i,j \in N^g, i \neq j} dist_{ij}}{\sum_{i,j \in N^g, i \neq j} \min(w_{ij}, h_{ij})}$$

Number of groups

Sum width of two macro

Sum height of two macro

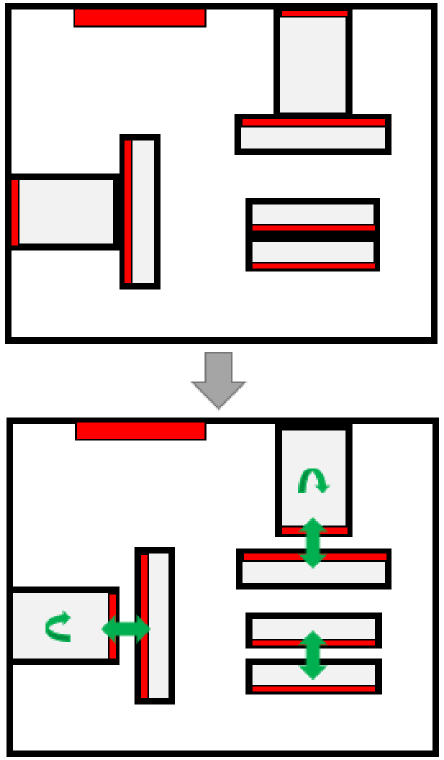


Effect of hierarchy cost

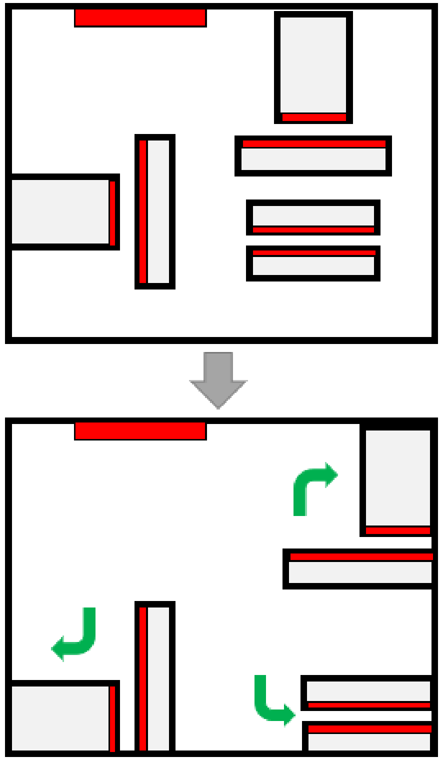
# Simulated Annealing-based Post Placement Engine

SA-based post placement is aim to achieve human-quality placement in terms of pin accessibility and dead-space minimization.

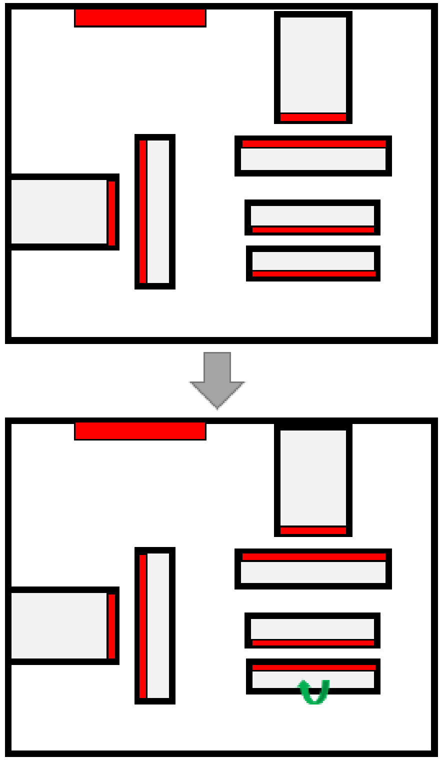
- Pin Constraints
- Orient the pins of edge macros inward.
- Maintain spacing between pins and other macros.



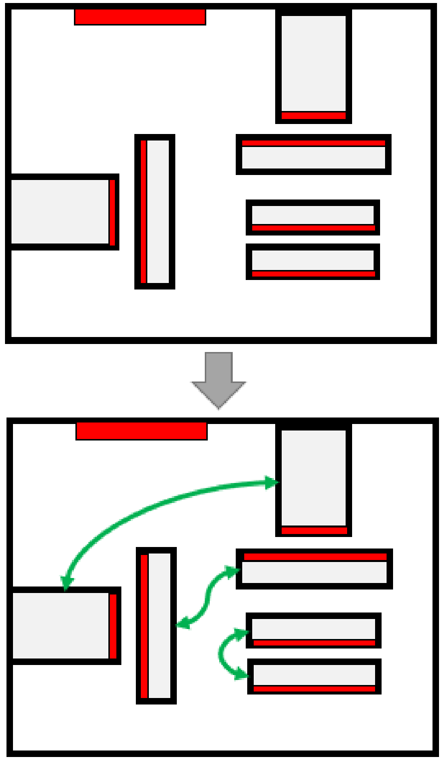
- Macro Action : Shift
  - Push macros towards the edge to reduce dead space.



- Macro Action : Flip
  - Flip or rotate macros to minimize wirelength.



- Macro Action : Swap
  - Modify macros of the same shape within the same group to reduce costs.

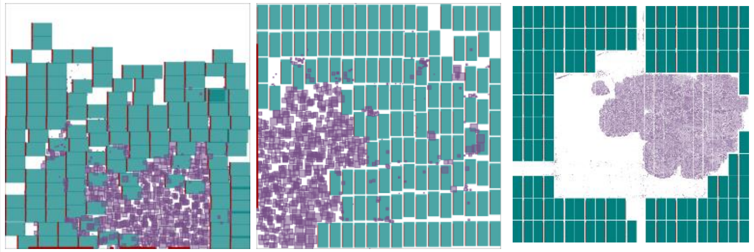


# Experiments

# Evaluation designs, flows, and settings

## Designs

| Designs | Netlist information | | | |
|---|---|---|---|---|
| | Core Size | # Macros | # IOs | # Clusters |
| Ariane (GCT) | 356.592 356.640 | 133 | 1231 | 799 |
| Ariane (TILOS) | 1347.1 1346.8 | 133 | 495 | 810 |
| Ariane (OURS) | 1445.9 1444.8 | 133 | 495 | 41 |

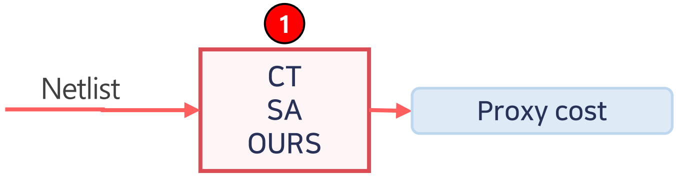


Ariane-GCT Ariane-TILOS Ariane-OURS

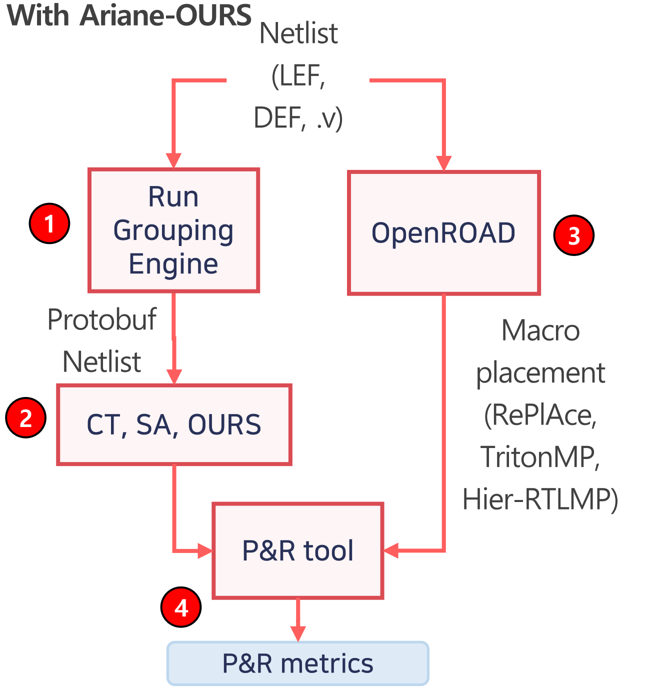
- We evaluate the framework using three netlists of Ariane CPU provided by [2] (Ariane-GCT), [10] (Ariane-TILOS), and a version we generated using NanGate45 standard-cell library (NG45)

## Flows

- **With Ariane-GCT and Ariane-TILOS**




- **With Ariane-OURS**



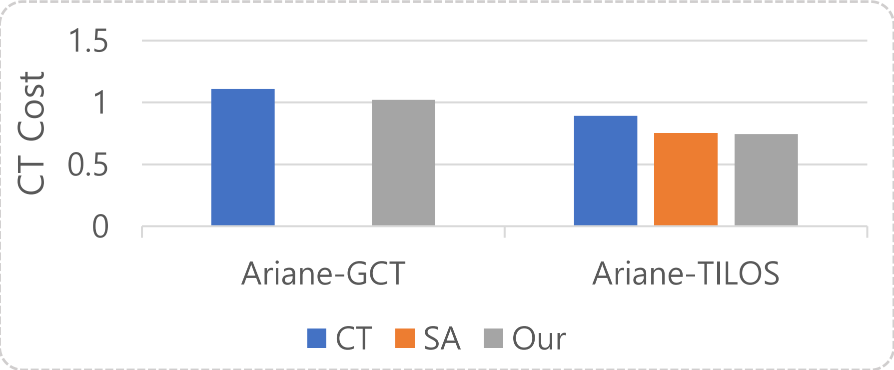

## Settings

| Designs | Model Configuration | | | |
|---|---|---|---|---|
| | Ori. Grid | Our Grid | # Nodes | # Edges |
| Ariane (GCT) | 35x33 | 12x18 | 1200 | 10000 |
| Ariane (TILOS) | 23x28 | 23x10 | 1200 | 12000 |
| Ariane (OURS) | - | 25x10 | 200 | 1100 |

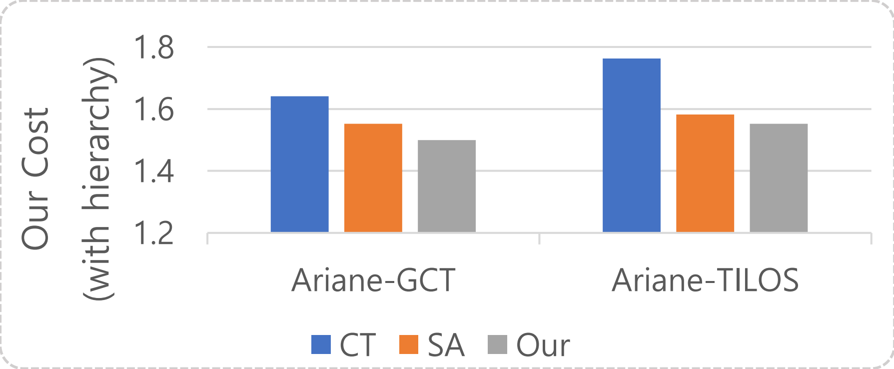
- **Infrastructure**:
  - A server with a 64-thread CPU, and an A5000 GPU with 24 GB of memory
  - Each run uses 25 collectors
- **Settings:**
  - We keep almost all training settings the same as the settings from [2] and [10].
  - The cost weights $\alpha$, $\beta$, $\gamma$, and ω were set to 5.0, 1.0, 0.5, and 0.1
  - We select the grid size ($Nr$ and $Nc$ ) relative to the chip canvas so that the smallest macro can fit inside a grid cell

[2] Circuit Training. https://github.com/google_research/circuit_training
[10] C. Cheng, A. Kahng, S. Kundu, et al. 2023. Assessment of Reinforcement Learning for Macro Placement. In Proc. ISPD. 158–166.

# 1.1 Evaluations Using Ariane-GCT and Ariane-TILOS netlist




1 Comparison with published results in [2][10]




2 Comparison by adding hierarchy cost

1 Our method can produce placements that show better proxy cost than those published in [2] and [10]

- Our method has 8% and 16.7% improvement (compared to CT) on Ariane-GCT and Ariane-TILOS, respectively

2 By adding hierarchy cost to the reward function:

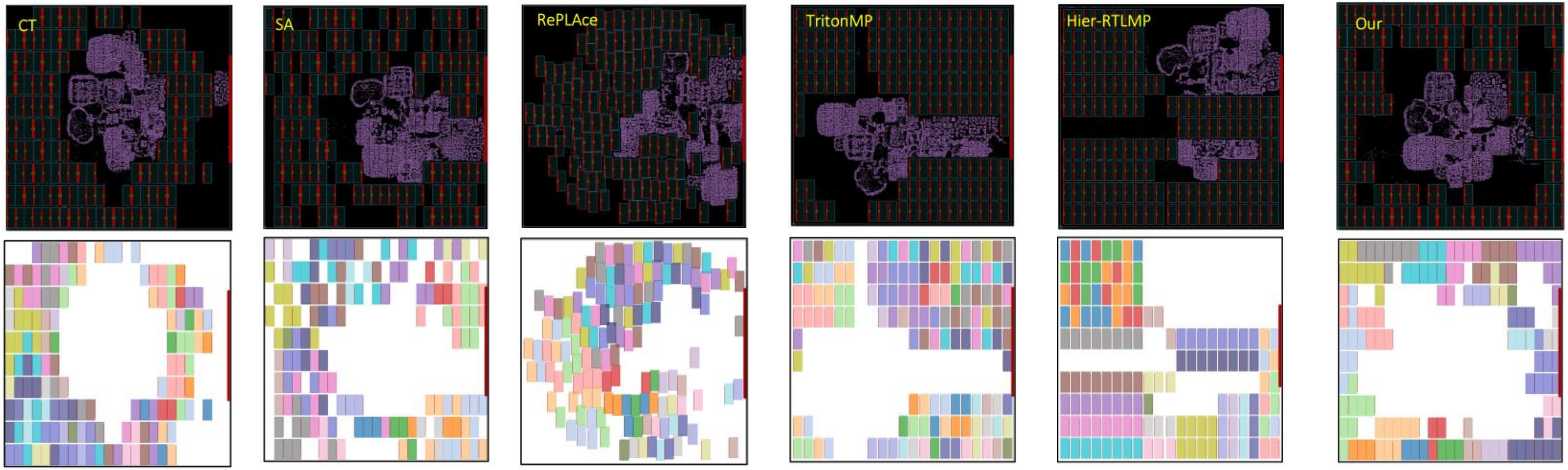
- Our method has 8.6% and 12% improvement (compared to CT) on Ariane-GCT and Ariane-TILOS, respectively

| | | CT metrics | | | | 1 | 2 | |
|---|---|---|---|---|---|---|---|---|
| Designs | Placer | WL Cost | Den. Cost | Cont. Cost | Hier. Cost | CT Cost | Our Cost | Inference time(h) |
| Ariane (GCT) | CT[2] | 0.1013 | 0.5502 | 0.9174 | - | 1.1102 | - | - |
| | $CT_{(12\times18)}$ | **0.0886** | 0.5345 | 0.8852 | 2.2115 | - | 1.6411 | 0.02 |
| | $SA_{(12\times18)}$ | 0.0963 | **0.5057** | 0.8446 | 1.4281 | - | 1.5523 | 14 |
| | $Our_{RL}$ | 0.0973 | 0.5088 | 0.8507 | 1.0571 | 1.0315 | 1.5264 | 0.02 |
| | $Our_{POST}$ | 0.0933 | 0.5070 | **0.8414** | **1.0565** | **1.0209** | **1.4997** | 0.1 |
| Ariane (TILOS) | CT[10] | 0.1060 | 0.5280 | 1.0470 | - | 0.8932 | - | - |
| | SA[10] | 0.0860 | 0.4990 | 0.8350 | - | 0.7533 | - | 12.5 |
| | $CT_{(23\times10)}$ | **0.0975** | 0.5860 | 0.7881 | 2.9580 | - | 1.7635 | 0.02 |
| | $SA_{(23\times10)}$ | 0.1061 | **0.5038** | 0.7761 | 1.5988 | - | 1.5820 | 10 |
| | $Our_{RL}$ | 0.1092 | 0.5121 | 0.7701 | **1.3207** | 0.7503 | 1.5752 | 0.02 |
| | $Our_{POST}$ | 0.1045 | 0.5156 | **0.7643** | 1.3211 | **0.7444** | **1.5522** | 0.1 |

[2] Circuit Training. https://github.com/google_research/circuit_training

[10] C. Cheng, A. Kahng, S. Kundu, et al. 2023. Assessment of Reinforcement Learning for Macro Placement. In Proc. ISPD. 158–166.

# 1.2 Evaluation using Our Generated Ariane Netlist



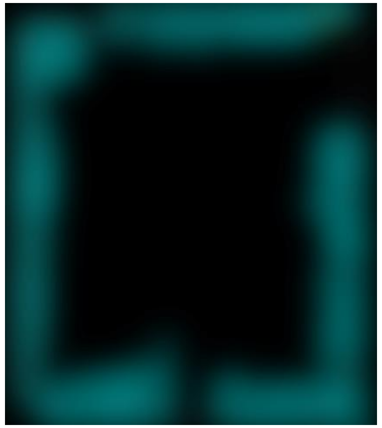

- In three out of four metrics, our framework has the best or second-best results compared to other placers.
- Our placer shows similarities to HierRTLMP in term of placing macros based on the design hierarchy, as well as similarities to both Hier-RTLMP and TritonMP in placing macros on the periphery

| Designs | Placer | P&R Metrics (post-route) Area ($mm^2$) | WNS (ns) | TNS (ns) | # DRC | Power (mW) | Proxy cost | Inference time (h) |
|---|---|---|---|---|---|---|---|---|
| Ariane (OURS) | $CT_{(25\times10)}$ | 1.2806 | -0.91 | -4833.9 | 9 | 585 | 1.8570 | 0.02 |
| | $SA_{(25\times10)}$ | 1.2850 | -0.93 | -5320.6 | 9 | 586 | 1.7879 | 14 |
| | RePLAce | 1.2812 | -1.04 | -5423.7 | 9 | 584 | 1.7244 | 1 |
| | TritonMP | 1.2839 | -0.89 | -5068.2 | 9 | 586 | 1.9621 | 1 |
| | Hier-RTLMP | 1.2823 | -0.84 | -4632.2 | 7 | 586 | 1.6482 | 8 |
| | Our | 1.2803 | -0.86 | -4731.0 | 6 | 586 | 1.5807 | 0.1 |



Agile SoDA

# 2. Evaluation of Industrial Designs

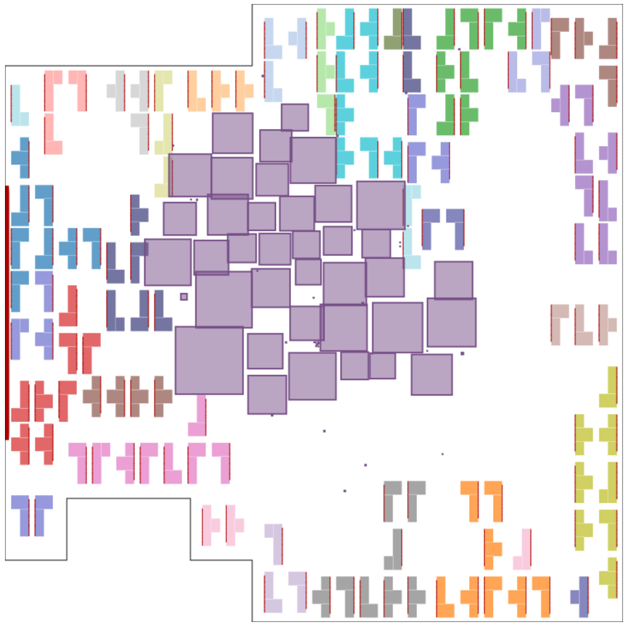


| Designs | # Macro | # Types | # IOs | # Cells | # Nets | Recti. Layout | Recti. Macros |
|---|---|---|---|---|---|---|---|
| ic1 | 89 | 59 | 1125 | 1.5M | 1.7M | ✓ | |
| ic2 | 169 | 97 | 630 | 3.8M | 4.3M | ✓ | |
| ic3 | 94 | 21 | 2207 | 1.8M | 1.8M | ✓ | ✓ |
| **Layout Metrics** | | | | | | | |
| Designs | Placer | Area ($mm^2$) | WNS (ns) | TNS (ns) | # DRC | Power (mW) | Run time(h) |
| ic1 | Human | 0.4550 | -0.6201 | -0.6201 | 2559 | 44.6 | weeks |
| | Comm | **0.4495** | **-0.6044** | **-0.6044** | **2491** | 46.8 | 0.5 |
| | Our | 0.4548 | -0.6178 | -0.6178 | 2695 | **43.7** | 14 |
| ic2 | Human | 1.0331 | -0.0709 | -376.68 | **6619** | 62.6 | weeks |
| | Comm | 1.0256 | -0.0739 | -302.11 | 23088 | **58.5** | 12 |
| | Our | **1.0206** | **-0.0698** | **-288.59** | 23542 | 59.8 | 28 |
| ic3 | Human | 5.7972 | -0.4193 | -1.4651 | **3924** | 284 | weeks |
| | Comm | 5.7965 | -0.4544 | -15.5075 | 5038 | 274 | 1.7 |
| | Our | **5.7961** | **-0.1402** | **-0.5792** | 4313 | **269** | 14 |

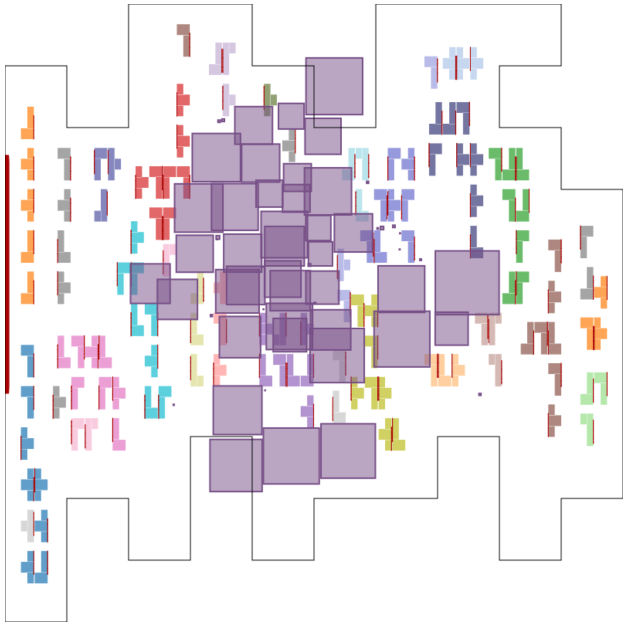
- We only applied reasonable efforts (no “benchmarking”), meaning we wanted to see if results were comparable, and not to try to prove if any such approach could “beat” the others.
- Our placer achieved PPA results that are better than those obtained by the designers within a few evaluations and are quite comparable to those achieved by the timing-driven placer from the P&R tool

# 3. Evaluation of Shape Generalization (#1)

- We create 100 random synthesized designs of Ariane-NG45 (80 for training and 20 for testing)
- We create 100 random synthesized designs of ASAP7 (for testing)
- We restricted the macro shapes to L, J and T patterns
- We avoided modifying macro shapes on their IO sides



100 random designs of Ariane-NG45

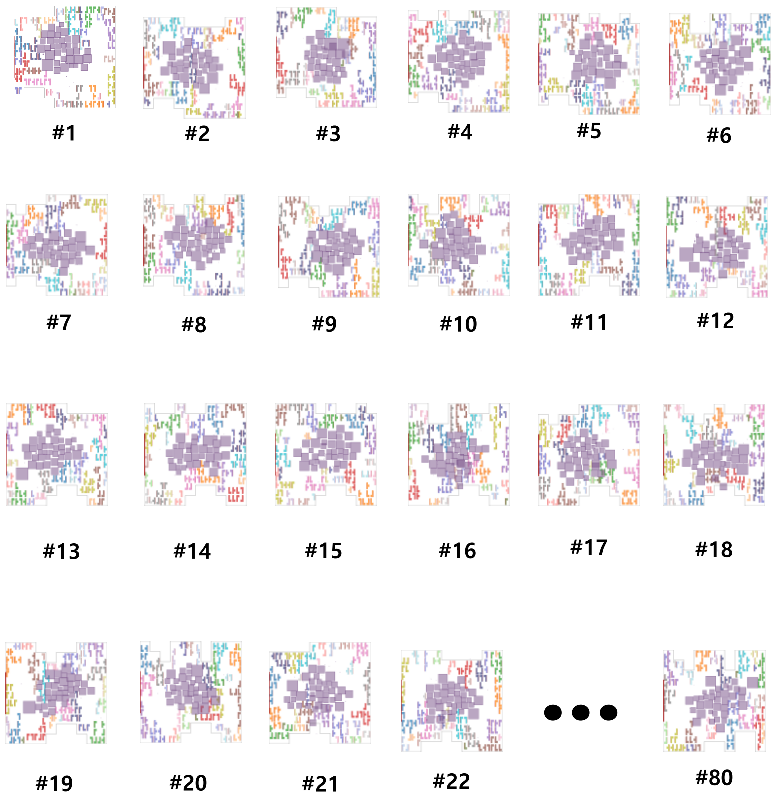


100 random designs of Ariane-ASAP7

# 3. Evaluation of Shape Generalization (#2)

The last experiment assessed the possible generalization of our model to designs containing rectilinear macros and areas.

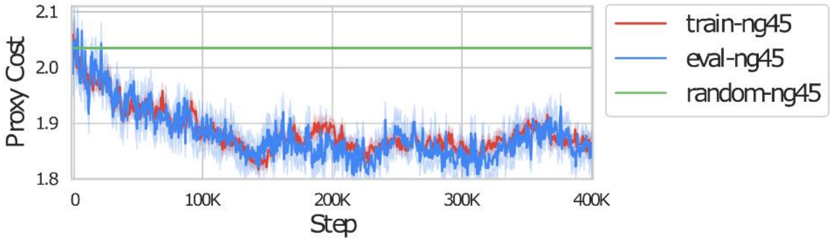
## Training on 80 synthesized Ariane-NG45




## Testing

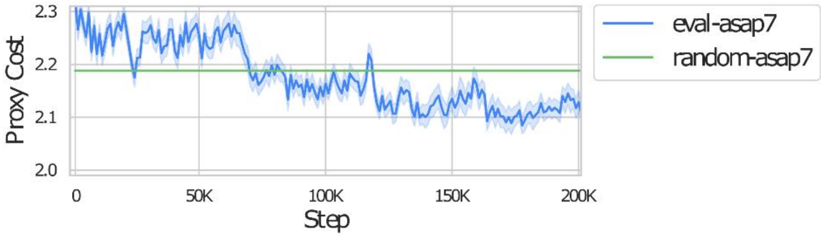
### 01 Testing on 20 synthesized Ariane-NG45

- The model is well trained to fit rectilinear designs from NG45



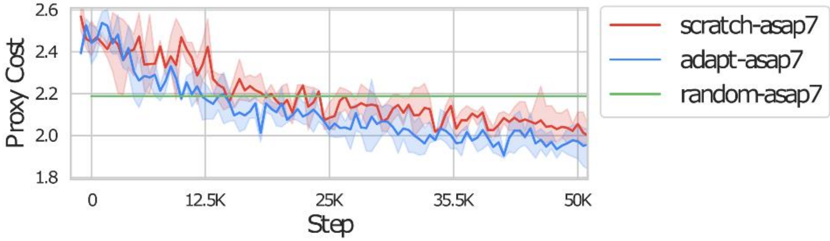

### 02 Testing on 100 synthesized Ariane-ASAP7

- The model-generated placements improved outperforming the random placements after 100K policy updates




### 03 Adaptation

- Adapting from a pre-trained model enabled the model to converge faster than training the model on that design from scratch






Agile SoDA

# 4. Runtime Analysis

| Designs | Inference Time (h) | Training Time (h) |
|---|---|---|
| A-GCT | 0.1 | 14 |
| A-TILOS | 0.1 | 10 |
| A-OURS | 0.1 | 14 |

Runtime

| Resource | GPU | CPU |
|---|---|---|
| GCT | 08 x A100s | 20 x 96vCPUs |
| TILOS | 08 x A100s | 02 x 96vCPUs |
| OURS | 01 x A5000 | 01 x 64vCPUs |

Training Resources

- Our learning-based placer only needs a few minutes to obtain a good placement (Inference Time)
- To generate a well-trained agent, we needs a few hours of training
- It's worth noting that with the same amount of training time, our placer consume fewer computing resources than other placers

# Conclusions

## Respects crucial human-like constraints

- Placement solution respects crucial human-like constraints
  - Design hierarchy
  - Peripheral bias

## Generalization

- This approach has the potential to generalize a learned model to various designs with rectilinear macros and areas.

## Reduce Training Resources

- We conducted on standard training machines.
- This can drive the research in RL-based placement towards efficiency and affordability.

# Demo page

https://anonymous.4open.science/r/rl4cad-AE0F

# Q & A

# Thank you!

Agile SoDA