DRC-Coder: Automated Design Rule Checking Code Generation

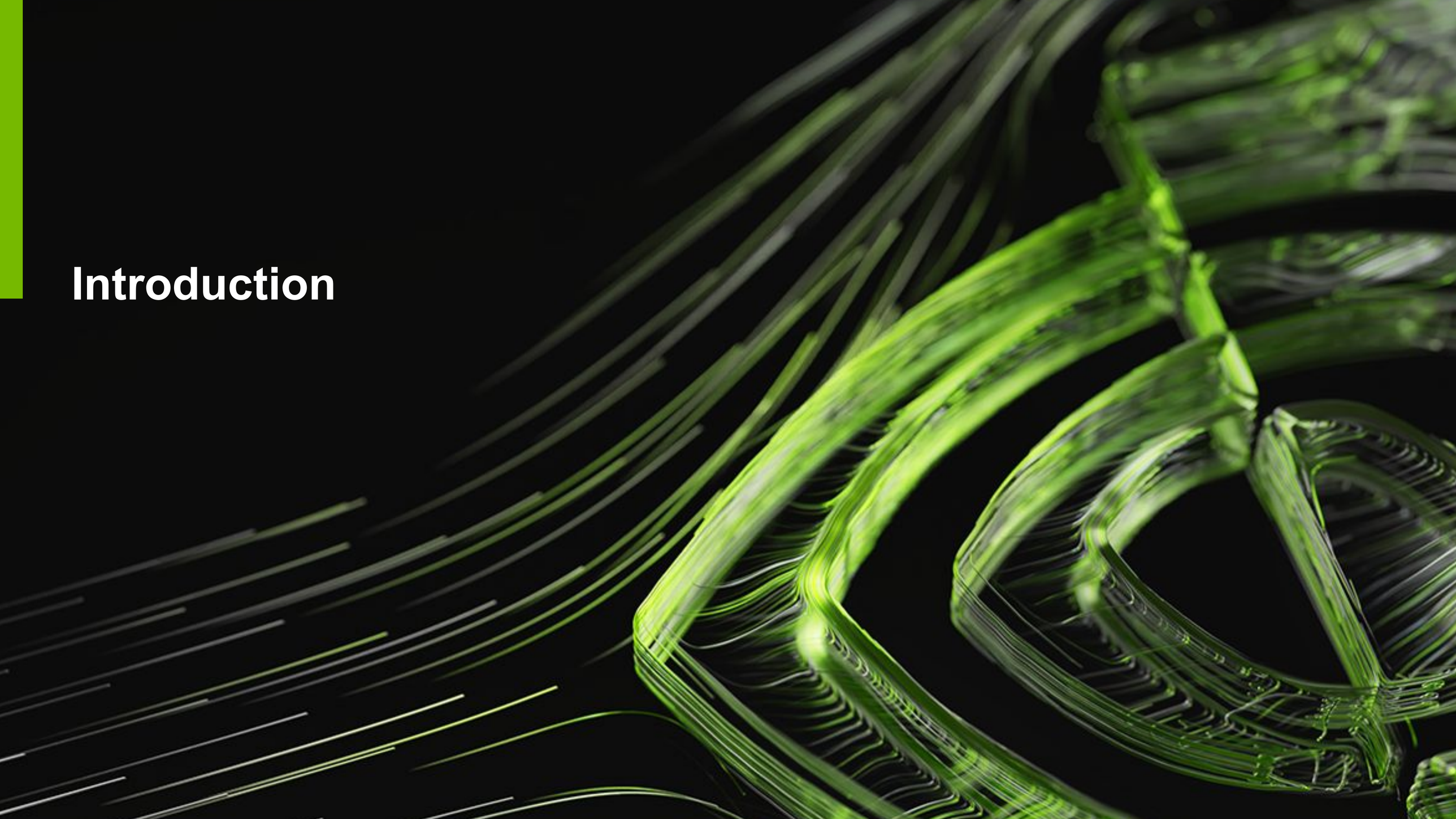
Chen-Chia Chang, Chia-Tung Ho, Yaguang Li, Yiran Chen, Haoxing Ren

Duke University, NVIDIA, NVIDIA Research

Presentation Outline

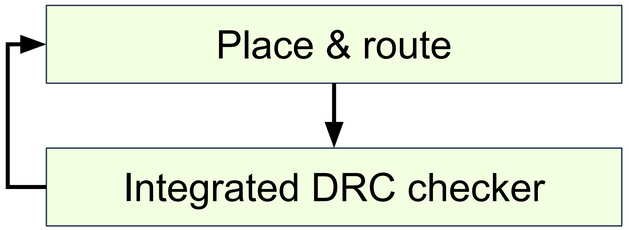
- Introduction
- DRC-Coder
- Experimental results
- Conclusion

Introduction

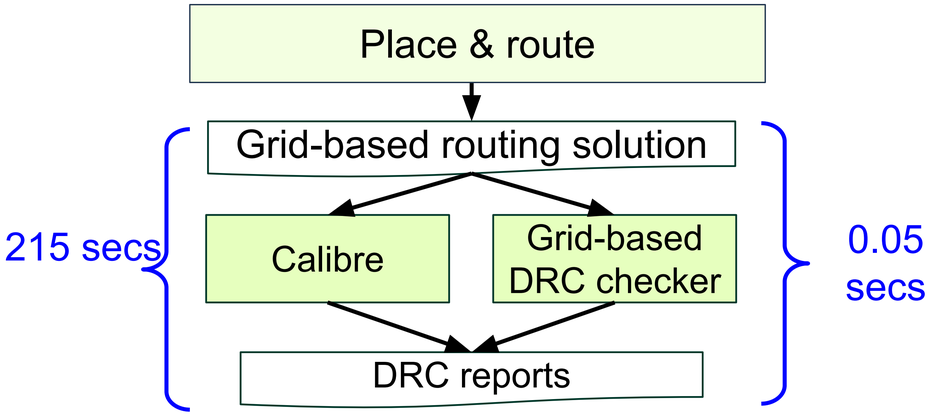


Design Rule Checking (DRC)

Physical design



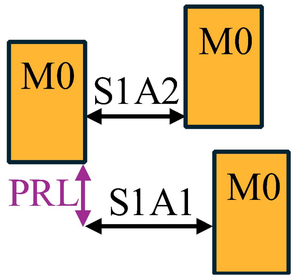
NVCell example



DRC checker development

Description for rule M0.S.1

Image:



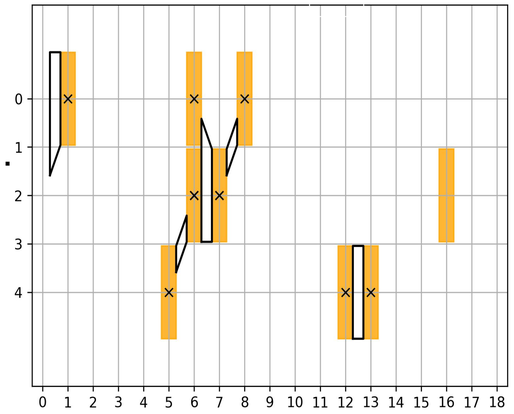
M0.S.1/M0.S.2

Text:

Space of M0, [PRL > -1], S1A1 > 1

Extracted layout & grid-based DRVs from commercial DRC tool report

Layout:
(8,0,M0,N5),
(5,4,M0,N9), …
DRVs:
((5,4,M0),
(6,2,M0)),
((1,0,M0)), …



Interpretation

Coding

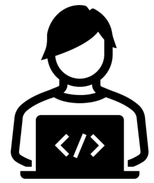
Checking alignment between checker & commercial tool report

Debugging

No

Yes

DRC checker



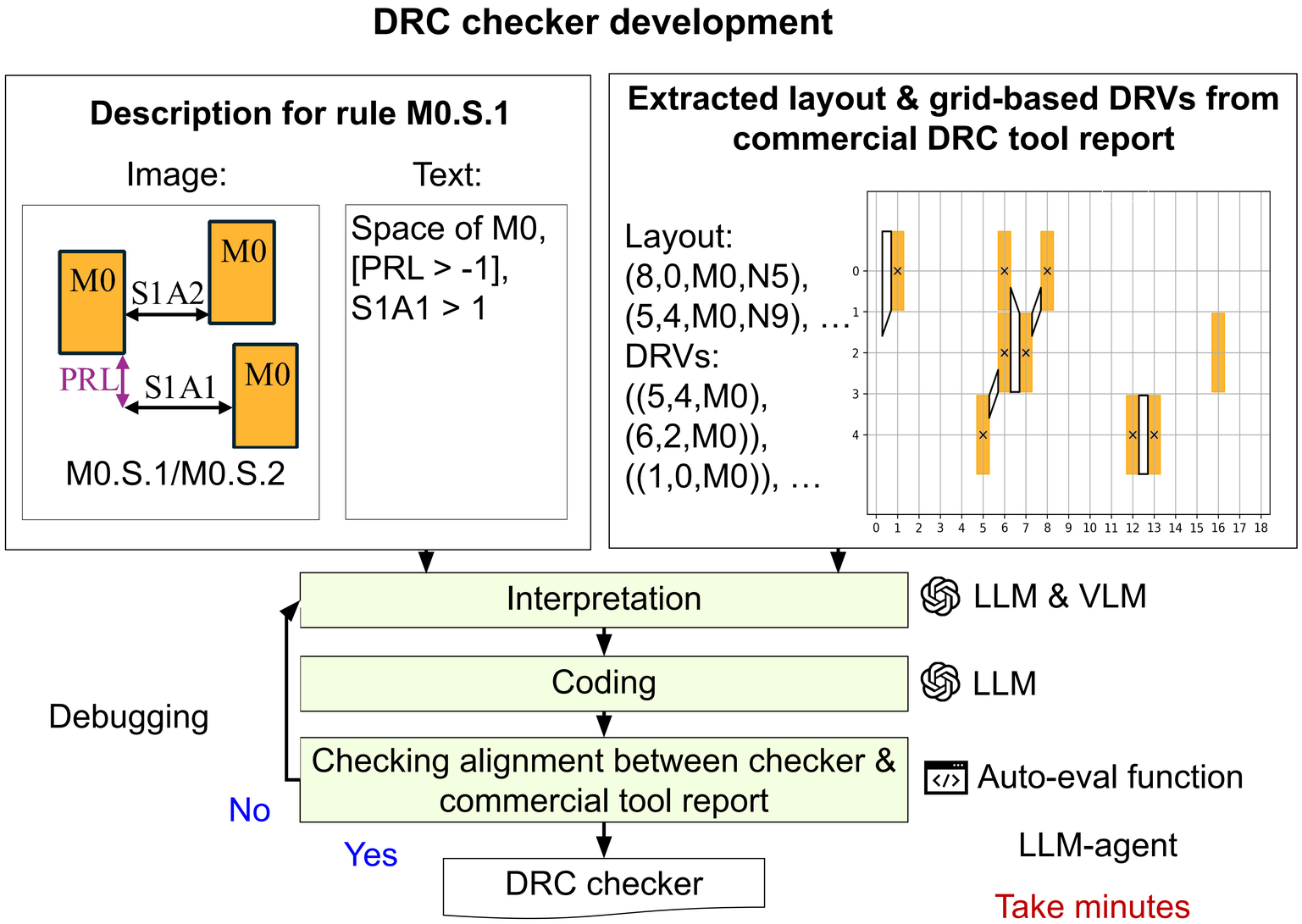
Human

Spend days ~ weeks

Automated DRC Code Generation

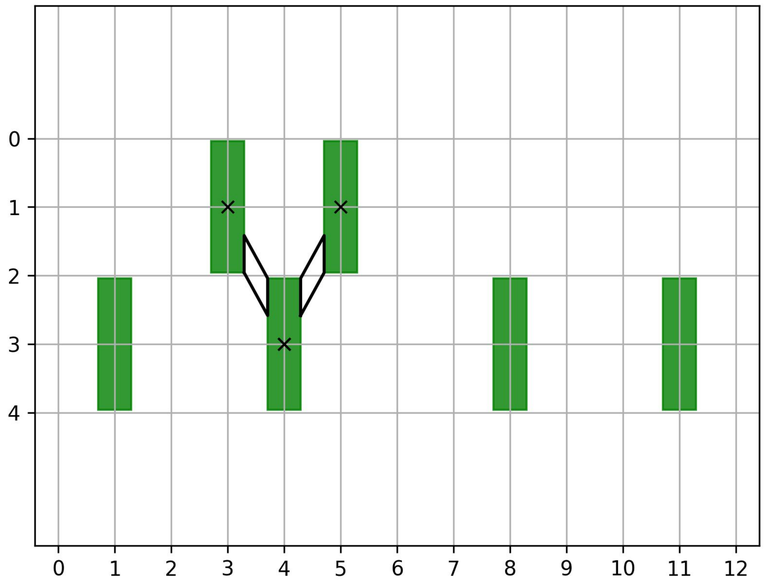
- LLM & VLM's ability
  - Code generation
  - Image interpretation

- DRC-Coder: Multi-agent with vision capability for DRC code generation
  - Reduce development time
  - Increase coding reliability
  - Enhance engineering productivity



Grid-based DRC Checker

- Input: Grid-based layout
- Output: DRVs



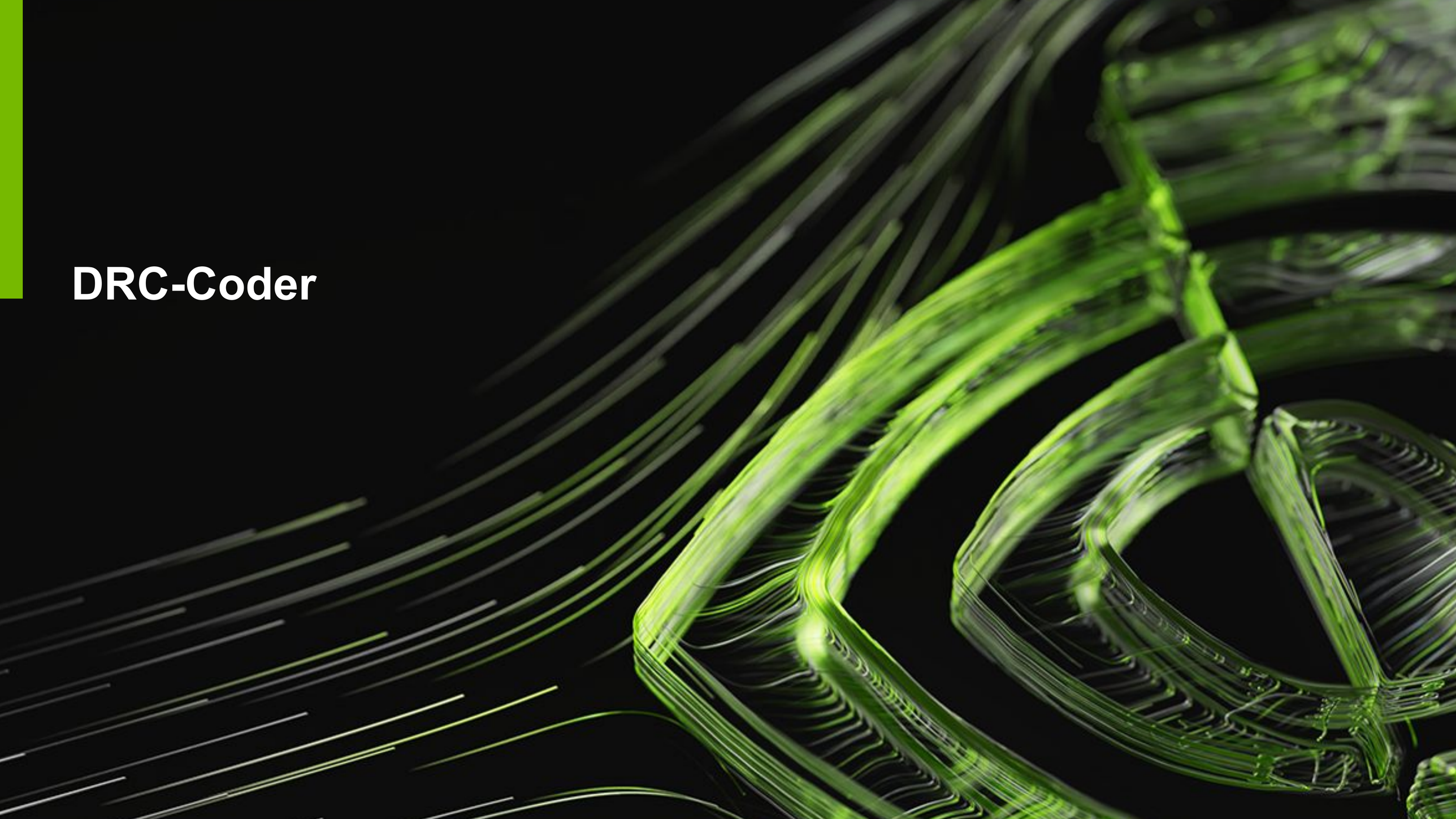
Layout:
(1 3), (3 1), (4 3),
(5 1), (8 3), (11 3)

→

DRVs:
DRV1 (3 1), (4 3)
DRV2 (4 3), (5,1)

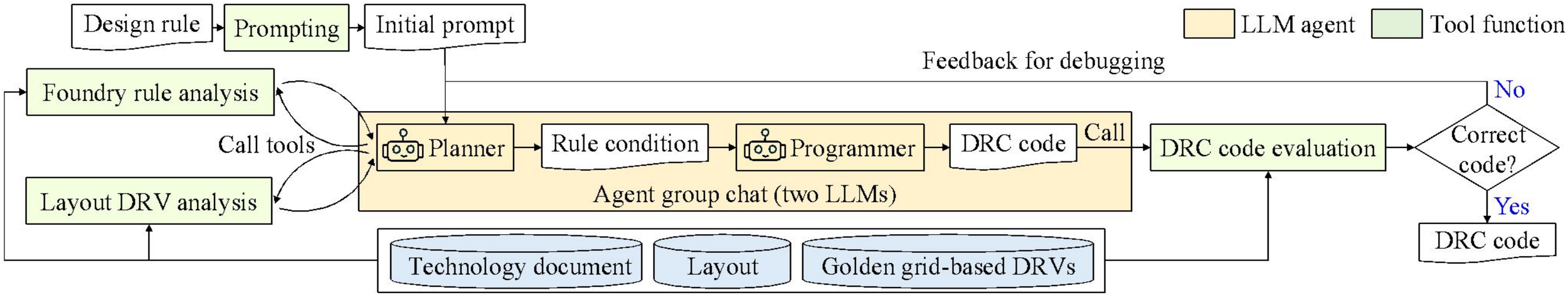
```python
def drc(layout_list, max_x, max_y):
    parsed_tuples = [(int(item.split(', ')[0][1:]), int(item.split(', ')[1]), item.split(', ')[2], item
    drvs = []

    # Check for DRVs
    for i in range(len(parsed_tuples)):
        for j in range(i + 1, len(parsed_tuples)):
            x1, y1, layer1, net1 = parsed_tuples[i]
            x2, y2, layer2, net2 = parsed_tuples[j]

            if layer1 == 'CM0B' and layer2 == 'CM0B':
                if y1 == y2 and abs(x1 - x2) <= 1:
                    drvs.append(f"({x1}, {y1}, {layer1}), ({x2}, {y2}, {layer2})")
                elif abs(y1 - y2) <= 2 and abs(x1 - x2) <= 1 and not (abs(y1 - y2) == 2 and x1 == x2):
                    drvs.append(f"({x1}, {y1}, {layer1}), ({x2}, {y2}, {layer2})")

    return drvs
```

DRC-Coder



DRC-Coder: Overview

- Interpretation and coding solved by multi-LLM
  - Planner: Summarize the design rule condition



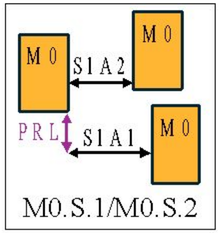
DRC-Coder

Planner

Foundry Rule Analysis

Input:
- Question from Planner: Explain design rule M0.S.1 in detail

Prompt to VLM:
Image: →



Text: You are an image-agent to help deriving DRC code.
Design rule description: Space of M0, [PRL > -1], S1A1 > 1
DRV could be the interaction between metals or between cell boundary.
You can summarize the design rule between polygons in the image.
Question of Planner: Explain the design rule M0.S.1 in detail.

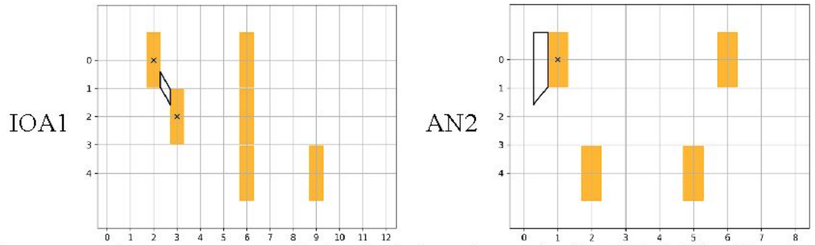
Output:
Identified Spacings in the Figure:
- S1A1: horizontal, PRL: vertical direction

DRC Conditions for Each Spacing
- Horizontal Direction Spacing S1A1:
  Check the spacing between M0 blocks where spacing is denoted by S1A1. Ensure spacing is greater than 1 grid unit.
- Vertical Parallel Run Length (PRL):
  Check parallel run lengths indicated by purple arrows. Ensure these metrics are greater than 0 grid units.

Layout DRV Analysis

Input:
- Question from Planner: Get the reason of DRVs
- List of cell names: [IOA1, AN2]

Prompt to VLM:
Image:



Text: You are an image-agent to help deriving the code for DRC. The black polygons are the DRV regions, the other polygons are layout components. The black cross is the grid-base DRV location corresponding to the components.
Question from Planner: Get the reason of DRVs

Output:
Cell IOA1
The DRV at (3, 2, M0): The black polygon is formed between the metals at (3, 2) and (2, 0), showing a diagonal interaction. This highlights a spacing issue between two metals, too close to each other.

Cell AN2
The DRV at (1, 0, M0): The black polygon highlights proximity of the metal at (1, 0) to the left boundary of the cell. This indicates the metal is too close to cell boundary, causing a violation.

DRC-Coder

Planner' Reply

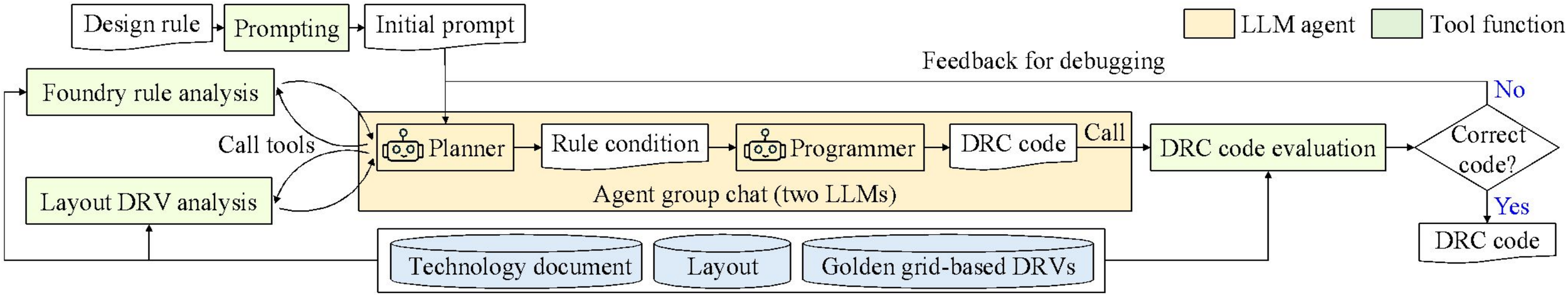
#Analysis of DRVs
Based on layout examples and the detailed explanation of the design rule, we can infer the conditions for DRVs:
Space between M0 metals in horizontal direction satisfies:
1. PRL > 2, 2. S1A1 > 1
Metals should not be too close to the x-boundary

#Plan to write DRC code
1. Boundary DRV Check:
   - Iterate M0 metals and check their proximity to the x boundary.

2. Spacing DRV Check:
   - Iterate pairs of M0 metals and check their horizontal spacing to see if the spacing is less than the PRL or S1A1.

DRC-Coder: Overview

- Interpretation and coding solved by multi-LLM
  - Planner: Summarize the design rule condition
  - Programmer: Write code based on design rule condition



DRC-Coder

Programmer

- DRC Code Evaluation Tool
  - Run DRC code on cell layouts
  - Compute overall performance
  - Report false negatives and false positives
    - Compute distances between DRV points
    - Report DRVs with unique conditions

Input:
- Generated DRC code: `def drc(layout, max_x, max_y): …`

Output:
Average performance of our generated code across 207 cell layouts:
Precision: 0.6570, Recall: 0.4894, F1 Score: 0.5540
The false negatives (missed DRV locations):
DRV #1 at (2, 2, M0) in cell SDFQ:
- Distance between DRV points (2, 2) and left boundary: **2 in x**.

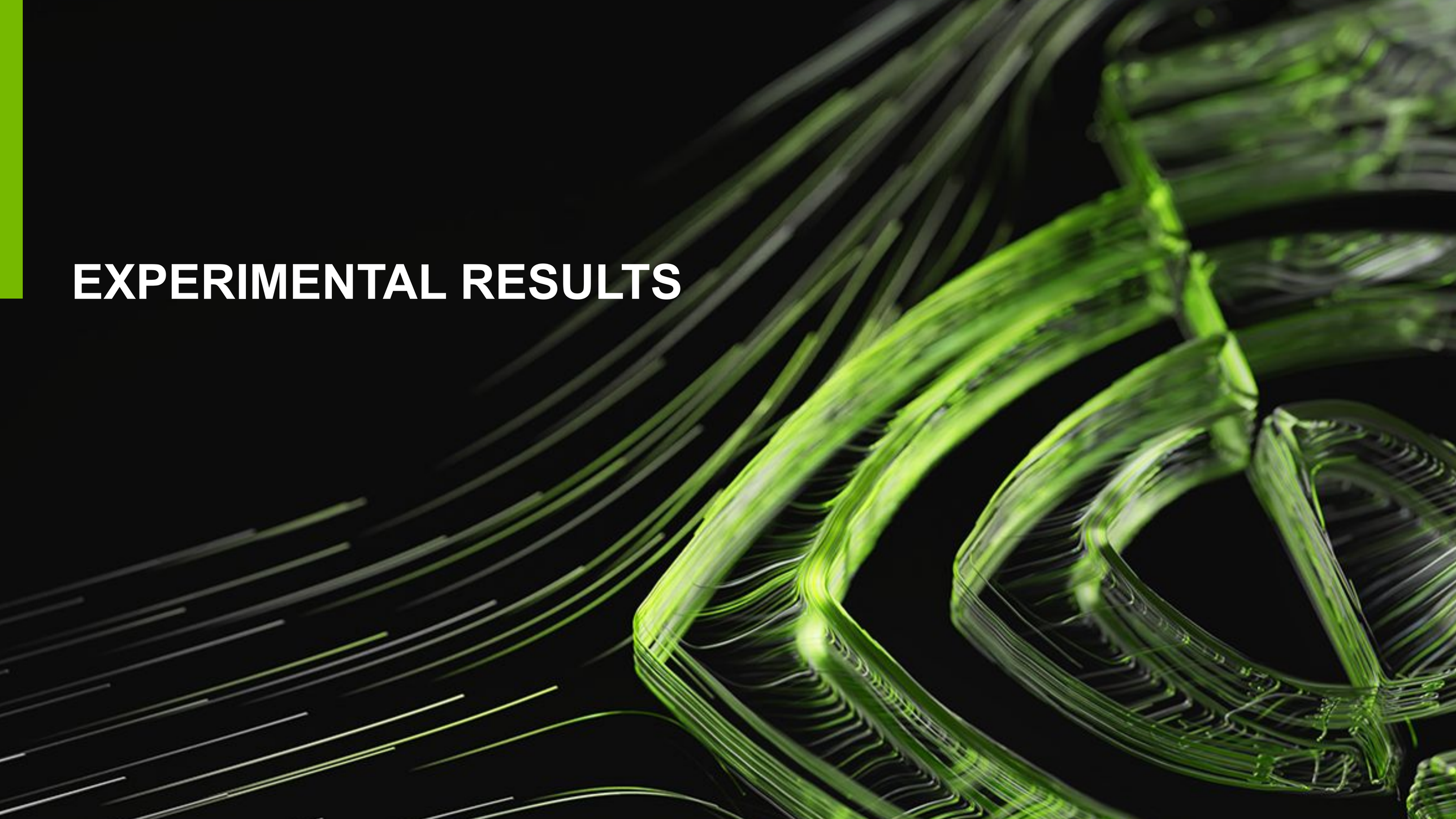
DRV #2 at (11, 2, M0) in cell XOR2:
- Distance between DRV points (11, 2) and right boundary: **2 in x**.

DRV #3 at (1, 2, M0) in cell XNR3:
- This DRV comes from the violation relation ((1, 0, M0), (1, 2, M0))
  Distance between DRV points (1, 0) and (1, 2): **0 in x and 2 in y**

###There is no false positives (incorrectly identified DRV locations)

We need to filter out the false negatives and false positives.
You can choose the following action:
(1) Use the x, y distance between DRVs to modify current DRC condition.
(2) If performance didn't improve compared to previous round, call Layout DRV Analysis by choosing two designs and specify your question in detail. You can ask what's the observation around detailed locations.

EXPERIMENTAL RESULTS



Setup

- Layout data generation with using NVCell
  - Number of layout: 207

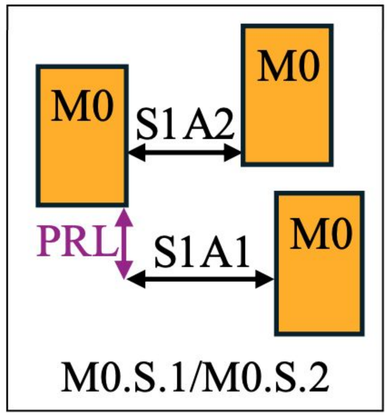
- Evaluation metrics (classification problem)
  - Precision (P): $TP/(TP+FP)$ Portion of actual DRV in predicted DRV
  - Recall (R): $TP/(TP+FN)$ Coverage of predicted DRV in actual DRV
  - F1: score (F): $2*(P*R)/(P+R)$ Balance of P and R

Actual DRV from Calibre

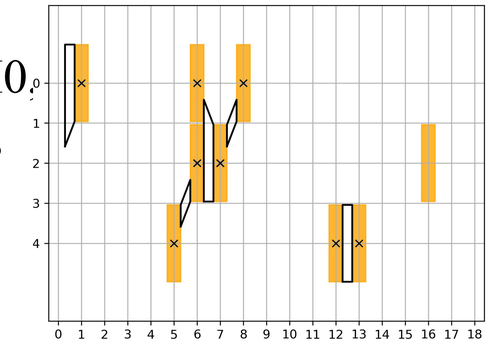
DRV from Generated Code	**DRV**	**No DRV**
DRV	TP	FP
No DRV	FN	TN

Design Rules

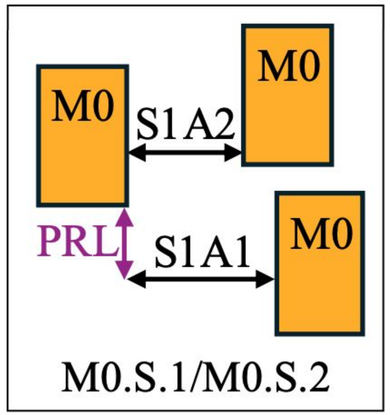
M0 Spacing Rule



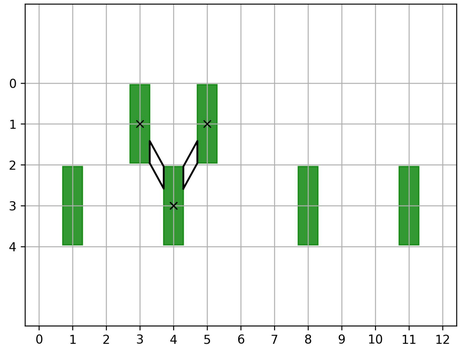
Space of M0,
[PRL > -1],
S1A1 > 1



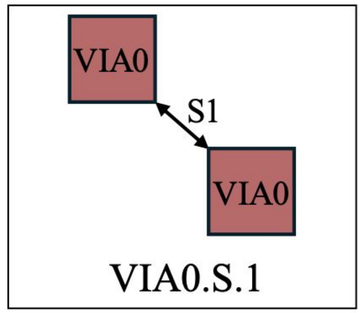
M1 Spacing Rule



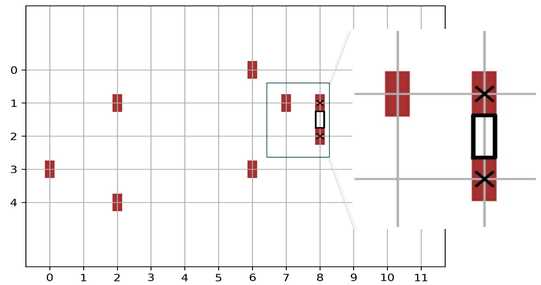
Space of M1,
S1A1 > 1



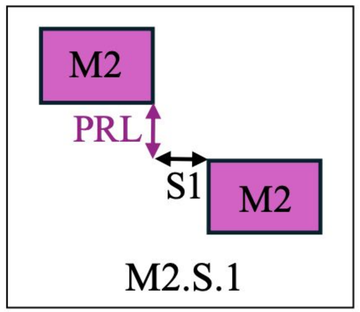
VIA0 & VIA1 Spacing Rule



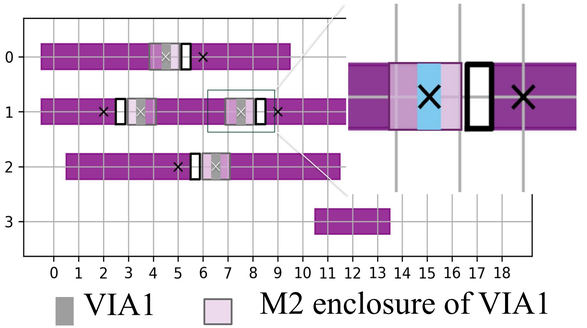
Space of
VIA0,
S1 > 1



Inter-layer (M2,VIA1) Spacing Rule



Space of M2 to
M2_line_end,
S1 > 1,
[PRL > -1]



Main Result

- Achieve perfect score (1.00)
- 37% higher F1 than standard prompting

LLM	Llama3						GPT-4o							
Rule	Standard Prompting			**DRC-Coder**			Standard Prompting			**DRC-Coder**				
	P	R	F	P	R	F	P	R	F	P	R	F	#iters	Runtime (s)
M0.S.1	0.841	0.710	0.745	**0.795**	**1.000**	**0.870**	0.789	0.527	0.620	**1.000**	**1.000**	**1.000**	**3**	**325**
M0.S.2	0.657	0.489	0.554	**0.696**	**0.817**	**0.722**	0.657	0.489	0.554	**1.000**	**1.000**	**1.000**	**2**	**121**
M1.S.1	0.646	0.403	0.483	**1.000**	**1.000**	**1.000**	0.645	0.402	0.482	**1.000**	**1.000**	**1.000**	**2**	**133**
M1.S.2	0.540	1.000	0.680	**1.000**	**1.000**	**1.000**	0.582	0.450	0.490	**1.000**	**1.000**	**1.000**	**3**	**354**
VIA0.S.1	0.000	0.000	0.000	**0.151**	**0.583**	**0.234**	0.659	1.000	0.769	**1.000**	**1.000**	**1.000**	**2**	**152**
M2.S.1	0.075	0.725	0.133	**0.149**	**1.000**	**0.255**	0.500	0.500	0.500	**1.000**	**1.000**	**1.000**	**1**	**45**
VIA1.S.1	0.220	1.000	0.356	**1.000**	**1.000**	**1.000**	1.000	1.000	1.000	**1.000**	**1.000**	**1.000**	**3**	**343**
Average	0.425	0.618	0.421	**0.684**	**0.914**	**0.726**	0.690	0.624	0.631	**1.000**	**1.000**	**1.000**	**2.3**	**210**

Ablation Study

- Evaluate the impact of VLM and multi-agent capabilities
  - Multi-Agent: 33% higher F1
  - VLM: 31% higher F1

LLM	GPT-4o											
Rule	Standard Prompting			Multi-agent w/o VLM			Single-agent w/ VLM			**DRC-Coder**		
	P	R	F	P	R	F	P	R	F	P	R	F
M0.S.1	0.789	0.527	0.620	0.951	0.657	0.747	0.944	0.965	0.946	**1.000**	**1.000**	**1.000**
M0.S.2	0.657	0.489	0.554	0.929	1.000	0.956	0.530	1.000	0.661	**1.000**	**1.000**	**1.000**
M1.S.1	0.645	0.402	0.482	1.000	1.000	1.000	0.660	1.000	0.769	**1.000**	**1.000**	**1.000**
M1.S.2	0.582	0.450	0.490	0.848	0.880	0.844	1.000	1.000	1.000	**1.000**	**1.000**	**1.000**
VIA0.S.1	0.659	1.000	0.769	1.000	1.000	1.000	1.000	1.000	1.000	**1.000**	**1.000**	**1.000**
M2.S.1	0.500	0.500	0.500	1.000	1.000	1.000	1.000	1.000	1.000	**1.000**	**1.000**	**1.000**
VIA1.S.1	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	**1.000**	**1.000**	**1.000**
Average	0.690	0.624	0.631	0.961	0.934	0.935	0.876	0.995	0.911	**1.000**	**1.000**	**1.000**

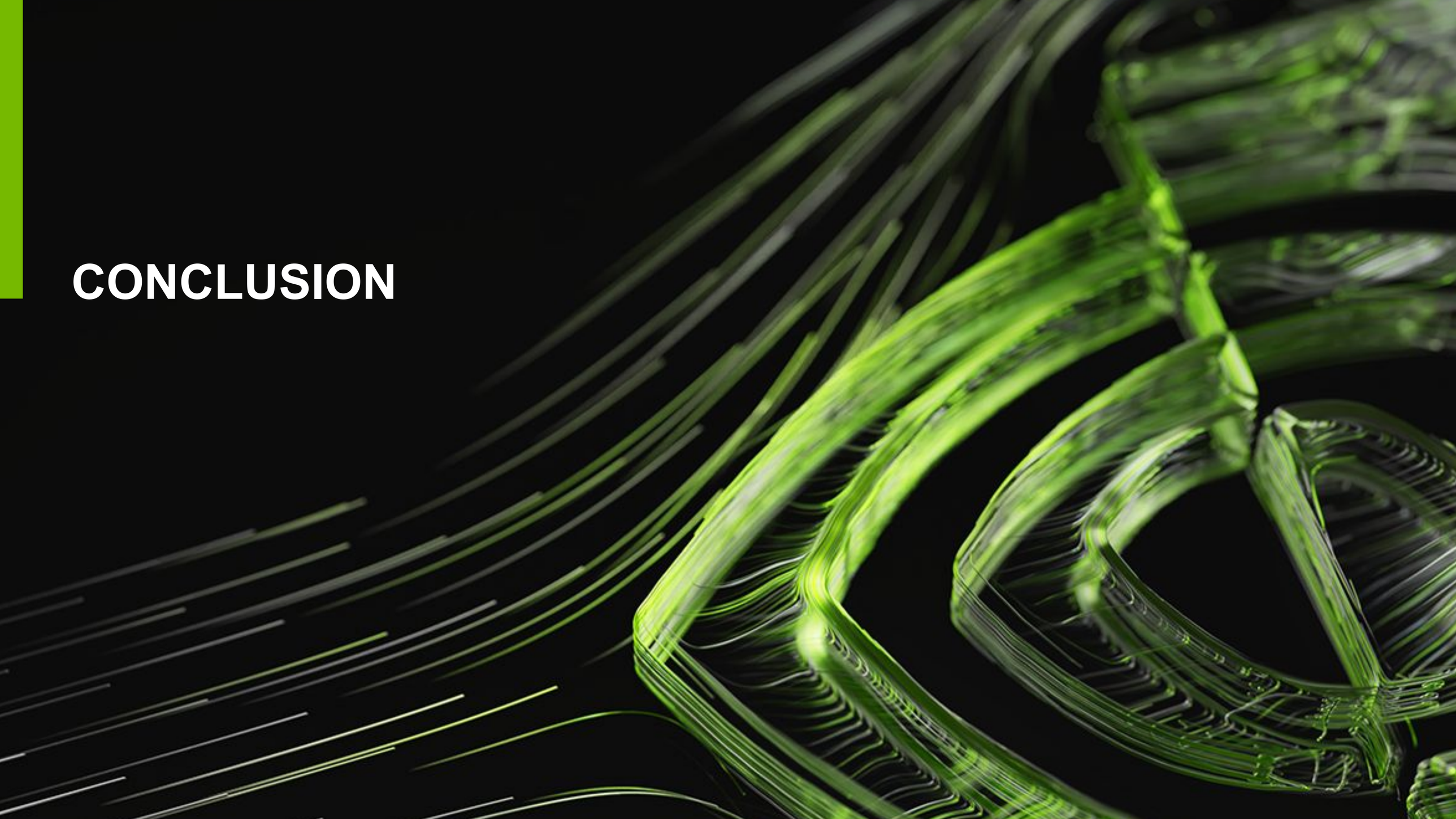
Case Study of Design Rule M0.S.1

- Solving flow (iterative improvement)
  1. foundry image & layout image explanation
  2. code evaluation tool F1: 0.747
  3. code evaluation tool F1: 0.966
  4. code evaluation tool F1: 0.747
  5. layout image explanation
  6. code evaluation tool F1: 0.917
  7. code evaluation tool F1: 1.000

Generated code

```python
def drc(layout_list, max_x, max_y):
    parsed_tuples = [(int(item.split(', ')[0][1:]), int(item.split(', ')[1]), item.split('

    drvs = []

    # Check for violations
    for i, (x1, y1, layer1, net1) in enumerate(parsed_tuples):
        if layer1 == "CM0A":
            # Check proximity to left x-boundary
            if x1 <= 1:
                drvs.append(f"({x1}, {y1}, {layer1})")

            # Check proximity to right x-boundary
            if x1 >= max_x - 1:
                drvs.append(f"({x1}, {y1}, {layer1})")

            # Check proximity to other metals
            for j, (x2, y2, layer2, net2) in enumerate(parsed_tuples):
                if i != j and layer2 == "CM0A":
                    if abs(x1 - x2) <= 1 and abs(y1 - y2) <= 2:
                        if not (abs(x1 - x2) == 0 and abs(y1 - y2) == 2):
                            drvs.append(f"({x1}, {y1}, {layer1}), ({x2}, {y2}, {layer2})")

    return drvs
```

CONCLUSION



Conclusion

- Develop DRC-Coder, a multi-agent framework with multi-modal vision capability
- DRC-Coder outperforms standard prompting techniques, achieving perfect F1 scores of 1.000 across all design rules considered in NVCell for a sub-3nm technology node
- DRC-Coder accelerates the development of DRC checker from days to minutes
- Future directions: Extend to a wide range of DRC-related applications
  - DRC-explanation chatbot
  - DRC-solving copilot


NVIDIA
Thank you!