Enhancing Reasoning to Adapt Large Language Models for Domain-Specific Applications

Bo Wen[1], Xin Zhang[1,2]
[1] IBM T. J. Watson Research Center, Yorktown Heights, NY, USA
[2] MIT-IBM Watson AI Lab, Cambridge, MA
Emails: bwen@us.ibm.com, xzhang@us.ibm.com

Abstract

This paper presents SOLOMON, a novel Neuro-inspired Large Language Model (LLM) Reasoning Network architecture that enhances the adaptability of foundation models for domain-specific applications. Through a case study in semiconductor layout design, we demonstrate how SOLOMON enables swift adaptation of general-purpose LLMs to specialized tasks by leveraging Prompt Engineering and In-Context Learning techniques. Our experiments reveal the challenges LLMs face in spatial reasoning and applying domain knowledge to practical problems. Results show that SOLOMON instances significantly outperform their baseline LLM counterparts and achieve performance comparable to state-of-the-art reasoning model, o1-preview. We discuss future research directions for developing more adaptive AI systems that can continually learn, adapt, and evolve in response to new information and changing requirements.

1 Introduction

The rapid advancements in large language models (LLMs) have revolutionized various aspects of artificial intelligence, enabling them to understand and generate human-like text with remarkable proficiency. However, adapting these general-purpose models to domain-specific tasks remains a significant challenge. In this paper, we introduce SOLOMON (System for Optimizing Language Outputs through Multi-agent Oversight Networks), a Neuro-inspired LLM Reasoning Network Architecture that leverages Prompt Engineering and In-Context Learning techniques, and demonstrate how SOLOMON can effectively adapt from its original design purpose in medical applications to a new domain: semiconductor layout design. Section 2 presents the SOLOMON architecture and highlights its design principles that contribute to enhanced adaptability.

To provide context for our experiment, we first examine how a designer might attempt to use ChatGPT (with GPT-4o mode) for a via connection design task in section 3. This exploration reveals a critical limitation: while LLMs can accurately recite textbook definitions of domain-specific concepts, they struggle to extract and apply expert knowledge to solve practical tasks. Human needs to translate high-level concepts into specific geometric requirements, which the LLM can then use to generate code for drawing shapes. This highlights the key challenge in adapting LLMs for domain-specific applications: their limited reasoning capabilities.

In section 4, we developed a set of 25 tasks ranging from basic geometric shapes to complex semiconductor structures, to evaluate our SOLOMON architecture against five different LLMs. These tasks assess spatial reasoning capabilities and adaptability across various complexity levels. Through these experiments, we demonstrate SOLOMON's superior performance compared to standalone LLMs, and reaching the level of state-of-the-art reasoning models like O1-preview.

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

Our findings emphasize the crucial role of reasoning in enhancing LLMs' adaptability to diverse domain applications. This study contributes to ongoing research in adaptive foundation models, providing insights into how to improve reasoning capabilities with inspiration from neuroscience.

2 Neuro-inspired LLM Reasoning Network Architecture

SOLOMON's architecture (Fig. 1) is inspired by two neuro-inspired theories: Brain-like AGI [1] and the Free Energy Principle (FEP) [2]. The former inspired us to use a pool of thoughts from multiple LLMs to discover the best reasoning plan. From the latter, we applied the FEP's main claim, *human attention focuses on minimizing the differences between goals and perceptions*, to select relevant information and avoid common pitfalls. The key components of SOLOMON are:

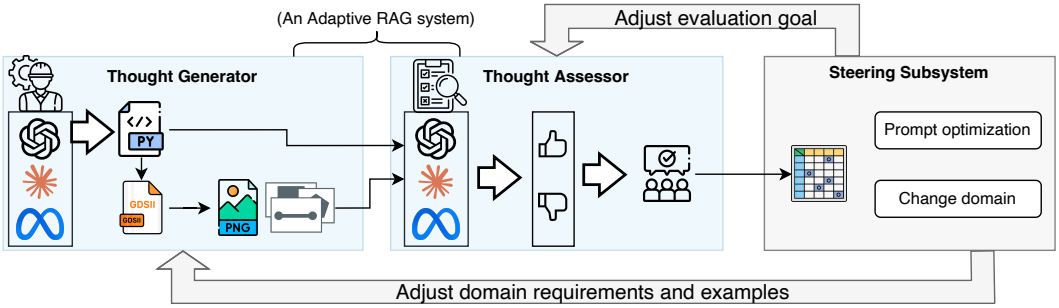


Figure 1: SOLOMON Architecture Diagram

Thought Generators: A diverse pool of LLMs generating thoughts for the target task. This component forms an efficient parallel search engine through the Tree-of-Thoughts [3, 4, 5, 6] and functions as an adaptive RAG system for the Thought Assessor. By pooling thoughts from multiple LLMs with distinct knowledge bases and reasoning abilities, it provides a more flexible and effective mechanism for sampling diverse ideas compared to common embedding-based RAG. This approach also mitigates biases inherent in single LLM knowledge bases. Noted that the individual LLMs in the Thought Generators can be further enhanced with proprietary knowledge through classic RAG techniques.

Thought Assessor: An LLM-based system that analyzes the proposed "Thoughts" to generate a refined output. It conducts in-context learning on the Thought Generators' output and follows the Free Energy Principle for goal-oriented assessments on consensus and differences. This approach enhances the LLM-as-a-Judge method [7, 8], enabling self-reflection [9] and guarding against hallucinations [10], thus improving AI safety and reliability.

Steering Subsystem: A human-operated component that controls the attention of the Thought Generators and Thought Assessor. It uses Prompt Engineering to modify the goals of other components, enabling swift adaptation to different domain requirements through goal-directed exploration of the search space. This enhances the system's versatility across various applications by simply adjusting the attention focus.

This architecture offers significant advantages over traditional fine-tuning approaches, eliminating the need for recurrent fine-tuning associated with upgrading underlying LLMs or updating domain-specific knowledge. Basing on Prompt Engineering techniques, SOLOMON enables building more flexible AI systems capable of addressing diverse specialized contexts.

3 Problem: Spatial Reasoning and Domain Knowledge Application

Layout design in semiconductor processes requires not only generating correct basic geometric shapes but also spatial reasoning to create proper "layouts" that meet specific requirements. Via connections, which create 3D electrical pathways between different chip layers, exemplify this challenge. While seemingly simple, typically consisting of circular vias and rectangular metal connections, they

demand precise positioning and sizing to ensure no short or open circuits when building the 2D layout into a 3D structure.

We conducted a series of tests by providing a sketch(image) together with different text prompts to ChatGPT (GPT-4o). The sketch are color-coded to represent different layers (e.g., yellow for via, blue for metal layer, red for contact pad) to help ChatGPT understand the spatial relationships. Figure 2 illustrates the sketch inputs and corresponding ChatGPT-generated outputs for each test case. In

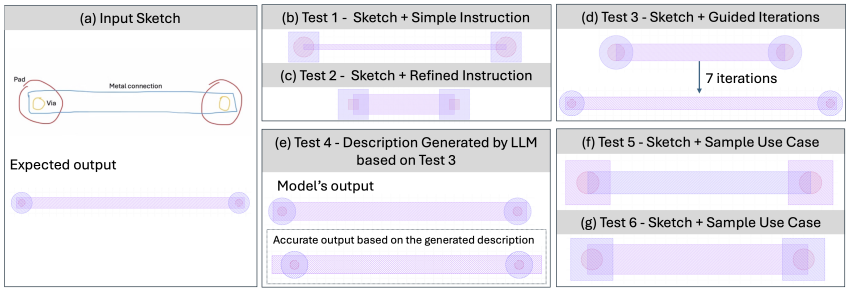


Figure 2: Sketch input and ChatGPT-generated outputs for the via connection experiment. The sketch depicts a desired layout with two vias connected by a metal layer and circular contact pads on top. The outputs show the progression of ChatGPT's understanding and refinement of the layout based on iterative feedback and context provided by the user.

Test 1, we provided the sketch (Figure 2(a)) with simple instructions. ChatGPT generated a runable code, but the metal layer width is narrower than the via. **Test 2** refined the instructions, but the output incorrectly used square vias [1], and failed to properly cover the vias with metal. In **Test 3**, we provided a more detailed description. After seven iterations of feedback, the LLM finally produced the correct layout (Figure 2(d)). See Appendix A.4 for detailed prompts and the iterative process.

Test 4 reversed the process by asking the LLM to create a detailed prompt based on the correct layout from **Test 3** (Appendix A.4). The LLM described each component's size and location in detail but hallucinated an additional requirement: *a 50-unit space between the vias and the edges of the metal connection*. This would result in the layout shown in the dotted rectangle in Figure 2(e), where the metal extends beyond the contact pad. Interestingly, when given this "wrong" prompt, GPT-4o ignored the added specification and produced a layout matching the original design, with the metal not extending beyond the pad. Code inspection revealed that the LLM used another requirement, *Leave a margin of 10 units between the edge of the metal and the pads*, to calculate the metal edge position in both x and y directions, although this statement was intended only for the y-direction margin. Using this version of prompt in the baseline evaluations (Section 4), o1-preview and Llama-3.1-405B each produced the "non-extending" version in one out of 5 runs, indicating some ambiguity in the specification.

To further test our hypothesis, we conducted **Tests 5** and **6**, removing numerical values from the prompt and incorporating domain-specific context (e.g., 3D packaging and Through-Silicon Vias). This approach, however, degraded LLM performance, revealing a critical limitation: while LLMs possess textbook knowledge of semiconductor concepts, they struggle to translate this into practical design requirements. For instance, LLMs failed to apply common engineering knowledge, such as using wider metal layers to connect vias or leaving margin space between components in different layers to account for layer misalignment.

This finding highlights a critical insight: to enhance the adaptability of LLM-based AI systems, simply increasing the model size to memorize more information is insufficient; instead, we should prioritize developing LLMs' reasoning capacity to effectively utilize their knowledge in practical problem-solving scenarios.

[1] square vias are difficult to fabricate in semiconductor etching process, so they are not used in practice

4 SOLOMON Performance and Comparison

To evaluate SOLOMON's effectiveness in enhancing spatial reasoning for semiconductor layout design, we created a dataset of 25 layout design tasks. These tasks were categorized into four groups: Basic Shapes 1 and 2 included simple geometric shapes such as circles, polygons and text, which serve as the building blocks for more complex layouts. The Advanced Shapes category involved more intricate designs, such as serpentine and spirals, to test the models' ability to handle complex geometries. Finally, the Complex Structures category included tasks that required the composition of multiple shapes to form functional layouts, such as a Dense Layer Diode (DLD) chip, MicrofluidicChip, and the ViaConnection test case. These tasks were designed to benchmark the AI systems' capability in generating layouts that are representative of real-world semiconductor design needs.

We provided task requirements with a system prompt asking the LLMs to use Chain-of-Thought to analyze the task and write Python code to create a GDSII output. The evaluation process involved running the generated code to produce GDSII files, which were then converted to PNG images. Human evaluators categorized the output into five categories: correct, scaling error, partially correct, shape error, and runtime error. Five LLMs (GPT-4o, Claude-3.5-Sonnet, Llama-3.1-70B, Llama-3.1-405B, and o1-preview) were used for the baseline experiment, with each task run 5 times per model. (See Appendix A.2 for details of prompts and example outputs.)

To evaluate SOLOMON, we utilized 20 thoughts generated by GPT-4o, Claude, and two Llama-3.1 models from the baseline experiment. We created four SOLOMON instances, each using one of these LLMs as a Thought Assessor, excluding o1-preview which served as our benchmark for state-of-the-art reasoning performance.

Figure 3 presents a performance comparison between the SOLOMON instances, their baseline counterparts, and the o1-preview model.

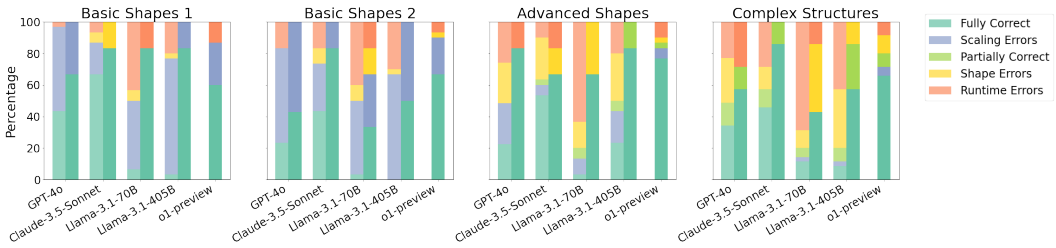


Figure 3: Performance comparison between SOLOMON instances, their baseline counterpart single LLMs, and o1-preview across different layout design task categories. Lighter colored bars on left represent baseline performance of individual LLMs, while darker bars on right show the performance of corresponding SOLOMON instances. O1-preview results serve as a benchmark for state-of-the-art reasoning performance.

The results demonstrate that the SOLOMON architecture significantly improves the performance of all four LLMs compared to their baseline. The most notable improvements are observed in the reduction of runtime errors, which can be attributed to the Thought Assessor seeing the error log of previous generated code and knowing what to avoid. This aligns with the design principle of the hierarchical, self-reflection mechanism, which aims to mitigate individual LLM's hallucination and blind-spot.

The second most notable problem in the baseline is scaling errors. We intentionally requested basic shapes to be drawn in millimeters to challenge the LLMs: they need to recall that the default unit in the gdspy library is micrometers. Sometimes LLMs simply failed to notice this and produced incorrect results. Additionally, each LLM seems to have bias when they hallucinate the default unit: Llama models perfers millimeters, Claude models sometimes recalls nanometers, and GPT-4o occasionally used meters. This issue was particularly problematic for Llama-3 models, when sometimes it correctly recalled the micrometer default but would insist that the user was wrong to request millimeters and proceed to draw without scaling, justifying it with comments like "not mm, as the GDSII format is in micrometers." Such "arrogant" behavior and misalignment with simple

instructions could be harmful for deploying LLMs as fully autonomous AI agents. A recent Nature paper [11] has also discussed similar observations.

The SOLOMON architecture improves performance across all models, including Llama-3. By incorporating diverse perspectives, it reduces stubbornness and increases accuracy. SOLOMON instances also show enhanced ability to handle shape errors and partial correctness issues, as the Thought Assessor can identify and correct errors related to arithmetic miscalculations or incorrect relative positioning of shapes. For more details, see Appendix A.3.

Comparing SOLOMON instances with o1-preview, we find that SOLOMON achieves comparable or superior results. All SOLOMON instances outperformed o1-preview in Basic Shape 1 categories, with the Claude-based SOLOMON surpassing o1-preview in 3 categories overall.

Interestingly, Llama-3 based SOLOMON instances also received significant performance boost, eventhough they don't receive the image inputs, suggesting that the thought assessment mechanism indeed works for more than just image understanding. Additionally, insufficient information linking images to corresponding code and error logs sometimes resulted in misinterpretation for GPT-4o and Claude.

Analysis of SOLOMON errors reveals that performance depends heavily on the quality and consensus of initial thoughts. Tasks with ambiguous requirements often leads to significant disagreement among initial thoughts, leading to confusion of Thought Assessor and degraded performance, see Appendix Table 9. These areas present opportunities for future improvements to the SOLOMON architecture.

5 Conclusion and Future Work

The introduction of the SOLOMON architecture significantly improved performance in semiconductor layout design tasks, particularly in reducing runtime errors and enhancing spatial reasoning capabilities. Our experiments demonstrated that SOLOMON instances outperformed their baseline LLM counterparts across various task categories, with some instances even surpassing the state-of-the-art o1-preview model in certain areas. This improvement validates the effectiveness of our neuro-inspired approach in enhancing LLMs' adaptability to domain-specific applications.

However, challenges remain in translating domain knowledge into practical design requirements. Our via connection experiment revealed that while LLMs can accurately recite textbook definitions of domain-specific concepts, they struggle to extract and apply expert knowledge to solve practical tasks. Investigating the potential of stacking multiple SOLOMON layers to form a hierarchical reasoning model capable of recalling and reasoning with domain knowledge for task-solving is one of our major future focus.

Other future research directions include: (1) Developing more comprehensive benchmark datasets for evaluating AI systems in layout design tasks. (2) Improving the linking between multimodal inputs (images and corresponding code+error) in the thoughts to enhance the Thought Assessor's interpretation abilities. (3) Exploring SOLOMON's performance when initial thoughts are of lower quality, and developing goal-oriented iterative learning mechanisms to improve thought quality through feedback loops. (4) Applying the SOLOMON architecture to a broader range of domain-specific tasks, such as power grid design and financial modeling.

In conclusion, while our results demonstrate the promise of LLMs as layout design copilots, further advancements in reasoning capabilities and domain knowledge application are necessary for their effective integration into semiconductor design processes and other specialized domains. The SOLOMON architecture represents a significant step towards creating more adaptive and capable AI systems for complex, domain-specific applications.

Open Source Code and Dataset: Visit our GitHub repository for the complete benchmark dataset of 25 tasks and LLM-calling code under the Apache 2.0 license at `https://github.com/wenboown/generative-ai-for-semiconductor-physical-design`. See Appendix A.1 for more details on Experiments Compute Resources requirements.

Acknowledgment: We thank Kuan Yu Hsieh for her valuable contribution in creating the dataset of 25 tasks with ground truth and her exploration work on the via connection test cases.

References

[1] Steven Byrnes. Intro to brain-like-agi safety. https://www.lesswrong.com/s/HzcM2dkCq7fwXBej8, 2022. A collection of blog posts, accessed on September 29, 2024.

[2] Thomas Parr, Giovanni Pezzulo, and Karl J. Friston. *Active Inference: The Free Energy Principle in Mind, Brain, and Behavior*. The MIT Press, 2022. In Special Collection: CogNet.

[3] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models, 2023.

[4] Yifan Zhang, Jingqin Yang, Yang Yuan, and Andrew Chi-Chih Yao. Cumulative reasoning with large language models, 2024.

[5] Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, and Torsten Hoefler. Graph of thoughts: Solving elaborate problems with large language models. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(16):17682–17690, March 2024.

[6] Maciej Besta, Florim Memedi, Zhenyu Zhang, Robert Gerstenberger, Guangyuan Piao, Nils Blach, Piotr Nyczyk, Marcin Copik, Grzegorz Kwaśniewski, Jürgen Müller, Lukas Gianinazzi, Ales Kubicek, Hubert Niewiadomski, Aidan O'Mahony, Onur Mutlu, and Torsten Hoefler. Demystifying chains, trees, and graphs of thoughts, 2024.

[7] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging llm-as-a-judge with mt-bench and chatbot arena, 2023.

[8] Yen-Ting Lin and Yun-Nung Chen. Llm-eval: Unified multi-dimensional automatic evaluation for open-domain conversations with large language models, 2023.

[9] Ziwei Ji, Tiezheng Yu, Yan Xu, Nayeon Lee, Etsuko Ishii, and Pascale Fung. Towards mitigating hallucination in large language models via self-reflection, 2023.

[10] Nuno M. Guerreiro, Elena Voita, and André F. T. Martins. Looking for a needle in a haystack: A comprehensive study of hallucinations in neural machine translation, 2023.

[11] Lexin Zhou, Wout Schellaert, Fernando Martínez-Plumed, Yael Moros-Daval, Cèsar Ferri, and José Hernández-Orallo. Larger and more instructable language models become less reliable. *Nature*, 2024.

A Appendix

A.1 Open Source Code and Dataset

To ensure reproducibility and facilitate further research, we release the complete benchmark dataset of 25 tasks and LLM-calling code under the Apache 2.0 license at `https://github.com/wenboown/generative-ai-for-semiconductor-physical-design`. This repository includes 5 runs of results (LLM answers, Python code, error logs, and PNGs) for each task in the baseline experiment for reproducibility. While the SOLOMON code is proprietary, its output results are included.

All LLM experiments were conducted by calling APIs: GPT-4o via OpenAI, Claude via Anthropic, and Llama 3s via IBM Watsonx. Total API costs are about $50 including re-runs of failed tasks and iterative testings. The local code (for calling APIs, collecting responses and saving to disk, running the LLM-generated code, and analysis) was run on a virtual machine with RHEL 8.0, equipped with a 32-core CPU and 256GB of memory.

The complete dataset, including prompts, ground truths, and LLM outputs, is available in our repository. This contains all materials needed to reproduce our baseline experiments and conduct further research.

A.2 Task Prompts and Baseline LLM Performance

The system prompt used for baseline experiment (thought generating) for all tasks was as follows:

```
You are an expert Python developer specialized in generating layout designs
in GDS (GDSII) format. Your task is to assist the user in creating Python
code that accurately draws layout designs while being mindful of the
geometric relationships and layout accuracy.

Write down your thinking step by step before you start coding:
1. Always start by understanding the overall design requirements provided
   by the user.
2. Break down the design into smaller components and define each geometric
   shape with precise coordinates.
3. Ensure that all shapes and elements maintain their correct geometric
   relationships, such as alignment, spacing, and proportional dimensions.
4. Validate each step of the design process to avoid errors and maintain
   accuracy.

Use the 'gdspy' library to generate the GDS layout:
1. Parse the user's design specifications.
2. Define the library and cell for the GDS layout.
3. Create each geometric element (e.g., rectangles, polygons) with precise
   coordinates.
4. Ensure elements are placed correctly and maintain their intended
   relationships.
5. Save the design to a GDS file.

Provide all the code in a single ```python ``` block to the user without
postamble. Do not include any other ``` block in your response to avoid
parsing error in following steps.

Be meticulous in your approach, and always consider the geometric
relationships and layout accuracy in every step of the design process.
```

Here are the complete list of all 25 task prompts (questions):

To aid human evaluators, we organized task prompts, ground truth images, and LLM output images from different runs in a tabular format. This presentation offers a clear view of various models' performance in generating GDSII layouts, enabling easy comparison between expected results and actual outputs from different LLMs and the SOLOMON system.

Table 1: Basic Shapes 1 Questions

Shape	Question
Circle	Write a Python code to generate GDSII for a circle on layer 0, radius = 10 mm, center at 0,0.
Donut	Generate a donut shape with 10 mm outer radius and 5 mm inner radius. Make the circle smoother by setting max distance between point 0.01mm.
Oval	Generate an oval with major axis of 20 mm, minor axis of 13 mm, on layer 0, center at 0,0.
Square	Generate a square with width 10 mm, put lower right corner of the square at 0,0.
Triangle	Generate a triangle with each edge 10 mm, center at 0,0.
Grid	Draw the GDSII for a grid: Grid on Layer 1, DATATYPE 4, 5 µm grid, and total width is 200 µm and height is 400 µm, placed at coordinates (100,800) nanometers.

Table 2: Basic Shapes 2 Questions

Shape	Question
Heptagon	Generate a Heptagon with each edge 10 mm, center at 0,0.
Octagon	Generate an Octagon with each edge 10 mm, center at 0,0.
Trapezoid	Generate a Trapezoid with upper edge 10 mm, lower edge 20 mm, height 8 mm, center at 0,0.
Hexagon	Generate a regular hexagon with each edge 10 mm, center at 0,0.
Pentagon	Generate a regular pentagon with each edge 10 mm, center at 0,0.
Text	Generate a GDS file with the text "Hello, GDS!" centered at (0,0), with a height of 5 mm, on layer 1.

Tables 6 through 9 at the Appendix's end showcase 5 representative experiment results. For complete results, see our GitHub repository.

A.3 Errors in Baseline Experiment

A.3.1 Scaling Errors

The default unit in the gdspy library is micrometers. We requested basic shapes to be drawn in millimeters to test whether LLMs could correctly handle this unit conversion. All LLMs struggled to various degrees:

(a) Some LLMs failed to pay attention to the requested unit (millimeters) and did not perform the necessary scaling.

(b) In some cases, LLMs paid attention to the requested unit but made incorrect assumptions about gdspy's default unit. We observed biased hallucinations: Llama models tended to assume millimeters,

Table 3: Advanced Shapes Questions

Shape	Question
Arrow	Generate an Arrow pointing to the right with length 10 mm, make the body 1/3 width of the head, start at 0,0.
SquareArray	Generate a square array with 5*5 mm square, for 10 columns and 10 rows, each 20 mm apart, the lower left corner of the upper right square is at 0,0.
Serpentine	Generate a serpentine pattern with a path width of 1 µm, 15 turns, each segment being 50 µm long and tall, starting at (0,0), on layer 2, datatype 6.
RoundedSquare	Draw a 10*10 mm square, and do corner rounding for each corner with r=1 mm.
Spiral	Generate a Parametric spiral with r(t) = e^(-0.1t), for 0 <= t <= 6pi, line width 1.
BasicLayout	1. Draw a rectangular active region with dimensions 10 µm x 5 µm. 2. Place a polysilicon gate that crosses the active region vertically at its center, with a width of 1 µm. 3. Add two square contact holes, each 1 µm x 1 µm, positioned 1 µm away from the gate on either side along the active region.

Table 4: Complex Structures Questions

Shape	Question
RectangleWithText	Generate a GDS with a 30*10 mm rectangle on layer 0 with a text "IBM Research" at the center of the rectangle. Put the text on layer 1.
MicrofluidicChip	Draw a design of a microfluidic chip. On layer 0, it is the bulk of the chip. It is a 30 * 20 mm rectangle. On layer 2 (via level), draw two circular vias, with 2 mm radius, and 20 mm apart horizontally. On layer 3 (channel level), draw a rectangular shaped channel (width = 1 mm) that connects the two vias at their center.
ViaConnection	Create a design with three layers: via layer (yellow), metal layer (blue), and pad layer (red). The via radius is 10 units, pad radius is 30 units, and metal connection width is 40 units with a total length of 600 units. Position the first via at (50, 150) and the second via at (550, 150). Ensure the metal connection fully covers the vias and leaves a margin of 10 units between the edge of the metal and the pads. Leave a space of 50 units between the vias and the edges of the metal connection.
FiducialCircle	Draw a 3.2 mm circle, with fiducial marks inside. The fiducial marks should be a "+" sign, with equal length and width. Each marker should be 200 um apart. There will be annotations next to each marker. Row: A -> Z, column: start from 1.
ComplexLayout	1. Draw three rectangular active regions with dimensions 20 µm x 5 µm, positioned horizontally with 5 µm spacing between them. 2. Create a complex polysilicon gate pattern consisting of multiple vertical and horizontal lines, with widths of 0.5 µm, forming a grid-like structure. 3. Add several contact holes (each 1 µm x 1 µm) positioned at the intersections of the polysilicon gate pattern and the active regions.
DLDChip	Draw a deterministic lateral displacement chip - include channel that can hold the array has gap size = 225 nm, circular pillar size = 400 nm, width = 30 pillars, row shift fraction = 0.1, add an inlet and outlet 40 µm diameter before and after the channel, use a 20*50 µm bus to connect the inlet and outlet to the channel.
FinFET	Draw a FinFET with the following specifications: - Fin width: 0.1 µm - Fin height: 0.2 µm - Fin length: 1.0 µm - Gate length: 0.1 µm - Source/drain length: 0.4 µm - Source/drain extension beyond the fin: 0.2 µm Use separate layers for the fin, gate, and source/drain regions.

Claude models sometimes defaulted to nanometers, while GPT-4o occasionally interpreted the default unit as meters.

(c) As mentioned in the main text, Llama-3 models were especially vulnerable to this issue. They sometimes assumed the user had made a mistake by requesting millimeters, and proceeded to draw in micrometers instead, justifying this choice with comments like "not mm, as the GDSII format is in micrometers".

A.3.2 Shape Errors

Incorrect shapes often resulted from LLMs making basic arithmetic errors. For instance, in the "Hexagon" task, Llama-3.1-405B once used an internal angle of 120 degrees, producing a triangle instead of a hexagon. However, in other runs, it correctly calculated the angle based on the number of edges. Many of these errors can be mitigated through Chain-of-Thought (CoT) prompting, which encourages the model to do calculations step-by-step.

A.3.3 Runtime Errors

This section provides a detailed breakdown of the errors encountered during the baseline experiment for each LLM. The most frequent error across all models was *AttributeError: module 'gdspy' has no attribute 'LayoutViewer'*, occurring 26 times (59.09%) with GPT-4o and 33 times (61.11%) with

Claude-3.5-Sonnet. This error was less common in other models, appearing only once each for Llama-3.1-70B and o1-preview, and not at all for Llama-3.1-405B.

The prevalence of this error indicates that GPT-4o and Claude-3.5-Sonnet attempted to provide GUI output, which was unavailable in the runtime environment. However, this issue stems from a lack of specification about the runtime environment in the prompt, rather than being entirely the LLMs' fault.

To ensure a fair comparison, we re-ran all generated code with 'LayoutViewer' lines commented out. The analysis presented in Figure 3 and the following breakdown reflect these adjusted results.

Other common errors included hallucinations of nonexistent 'gdspy' functions or methods, resulting in various 'AttributeErrors' (e.g., ''CrossSection'', ''Circular'', ''Ellipse'') and 'TypeErrors'. Some errors were due to spelling mistakes, such as misspelling *gdspy.Text* as *gdspy.text*.

The subsequent analysis presents a detailed error breakdown for each LLM, ranked by ascending number of errors.

o1-preview Total errors: 12

The main errors for o1-preview included:

- TypeError: GdsLibrary.write_gds() got an unexpected keyword argument 'unit' (16.67%)
- SyntaxError: invalid syntax (16.67%)
- Various TypeErrors and AttributeErrors related to unexpected keyword arguments or missing attributes (66.66%)

GPT-4o Total errors: 18

The most common errors for GPT-4o were:

- TypeError related to unexpected keyword arguments (27.78%)
- SyntaxError: invalid syntax (16.67%)
- TypeError: 'float' object cannot be interpreted as an integer (11.11%)

Other errors included various AttributeErrors, IndexErrors, and ValueErrors, each occurring once or twice.

Claude-3-5-sonnet Total errors: 21

The most frequent error for Claude-3-5-sonnet was:

- TypeError: Text.__init__() got an unexpected keyword argument 'anchor' (38.10%)

Other errors included:

- TypeError: Path.__init__() got an unexpected keyword argument 'layer' (9.52%)
- Various TypeErrors, ValueErrors, and AttributeErrors, each occurring once (52.38%)

Llama-3-405b Total errors: 36

The most frequent errors for Llama-3-405b were:

- TypeError: 'int' object is not subscriptable (8.33%)
- SyntaxError: invalid syntax (8.33%)
- Various TypeErrors related to unexpected keyword arguments or multiple values for arguments (16.68%)

This model also encountered several ValueErrors and AttributeErrors.

Llama-3-70b Total errors: 68

The most prevalent error for Llama-3-70b was:

- AttributeError: module 'gdspy' has no attribute 'Library'. Did you mean: 'library'? (36.76%)

Other common errors included:

- TypeError: GdsLibrary.write_gds() got an unexpected keyword argument 'unit' (7.35%)
- SyntaxError related to assignment (5.88%)
- Various TypeErrors and AttributeErrors related to unexpected keyword arguments or missing attributes (25.00%)

These error patterns suggest that all models struggled with correctly using the gdspy library, often attempting to use non-existent attributes or passing incorrect arguments to functions. Syntax errors were also common across models, indicating issues with code structure and Python syntax.

A.3.4 Inefficient Code

In the DLDChip task, which involves creating a dense array of identical shapes, the Llama-3.1-405B model generated code that created a large number of objects and performed numerous boolean operations. This led to high memory usage and extended execution time, requiring the code to be terminated after approximately 15 minutes of runtime.

A.3.5 Ambiguous Instructions

In some cases, we observed that the LLM results mainly fell into two categories. After inspecting the prompts, we found that the instructions could be interpreted in two ways. In these cases, we counted both types of results as correct. However, when implementing a copilot, the agent should ask for clarification if the instructions are ambiguous.

A.4 Via Connection Test Cases

Figure 4 shows the sketch used in all via connection test cases discussed in Section 3.

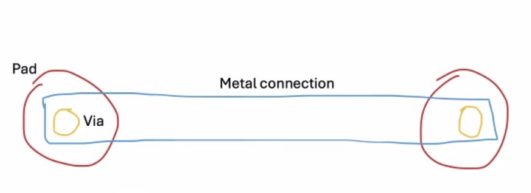


Figure 4: Sketch used for via connection test cases

A.4.1 Prompts for Via Connection Tests

Test 1: "I have a sketch idea that i want to draw in GDSII, generate the python code for this design. each color represents an individual layer. We want to use a metal to connect two vias and put a pad on top of each via"

Test 2: "I have a sketch idea that i want to draw in GDSII, generate the python code for this design. each color represents an individual layer. We want to have two vias near each end on a piece of metal. And a pad on top of the metal."

Test 3: "I have a sketch idea that i want to draw in GDSII, generate the python code for this design. each color represents an individual layer. we want use to connect two vias using a piece of metal and put a circular padding on top of each via"

Figure 5 illustrates the iterative prompting process for Test 3, as discussed in Section 3.

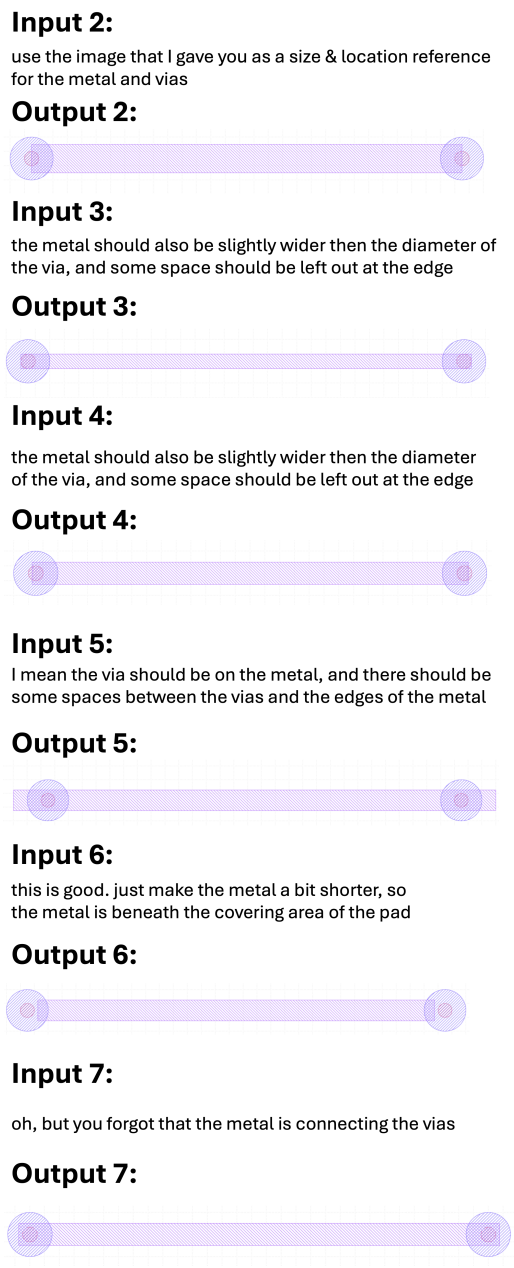


Figure 5: Test 3 - Iterations to guide the model to generate desired output

Test 4 (Generated by LLM based on the final output in Test 3):

```
Layers and Colors:
The design consists of three layers: via layer (yellow),
metal layer (blue), and pad layer (red).
Dimensions:
Via: The radius of each via is 10 units.
Pad: The radius of each pad is 30 units.
Metal Connection: The width of the metal connection is 40 units,
and the total length is 600 units.
Positions:
The first via is positioned at coordinates (50, 150).
```

```
The second via is positioned at coordinates (550, 150).
Connections and Coverage:
The metal connection should fully cover the vias, extending slightly
beyond their edges.
Ensure the metal connection is slightly wider than the via diameter
to provide full coverage.
Spacing and Margins:
Leave a margin of 10 units between the edge of the metal and the pads.
Ensure there is a space of 50 units between the vias and the edges
of the metal connection.
Additional Requirements:
The metal connection should be shorter than the total length to fit
beneath the covering area of the pads, leaving some space at the edges.
By providing detailed information like this, you can ensure that
the design is accurately reproduced.
If you have any specific design rules or preferences, make sure to
include those as well.
```

Test 5: "I have sketched a design for 3d packaging, where we have a metal connecting two TSVs , please generate the python code to draw a GDSII for this design."

Test 6: "I have sketched a design for 3d packaging, where we have a metal connecting two TSVs , please generate the python code to draw a GDSII based on the sketch. each color represents an individual layer. The metal connection should fully cover the vias, extending slightly beyond their edges. Ensure the metal connection is slightly wider than the via diameter to provide full coverage."

Table 5: Oval Task Question: Generate an oval with major axis of 20 mm, minor axis of 13 mm, on layer 0, center at 0,0.

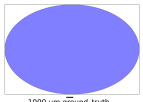
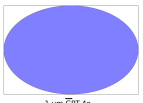
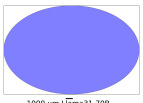
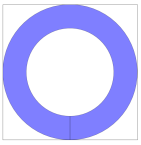
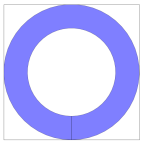
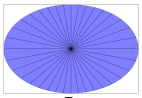
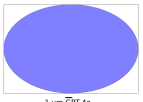
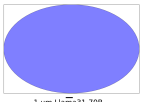
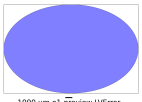
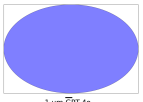
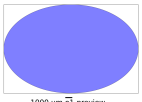
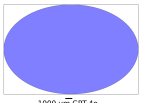
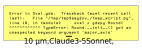
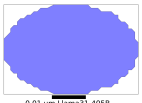
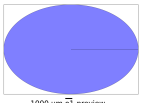
Ground Truth	GPT-4o	Claude-3.5	Llama-3-70B	Llama-3-405B	o1-preview
 1000 µm,ground_truth,					
SOLOMON	 1 µm,GPT-4o,	1000 µm,Claude3-5Sonnet,	 1000 µm,Llama31-70B,	1000 µm,Llama31-405B,	
Single LLM Baseline Run 1	10 µm,GPT-4o,	 1000 µm,Claude3-5Sonnet,LVError	 1 µm,Llama31-70B,	10 µm,Llama31-405B,	10 µm,o1-preview,
Single LLM Baseline Run 2	1 µm,GPT-4o,	1000 µm,Claude3-5Sonnet,	10 µm,Llama31-70B,	1 µm,Llama31-405B,	 1000 µm,o1-preview,
Single LLM Baseline Run 3	 1 µm,GPT-4o,	1000 µm,Claude3-5Sonnet,	 1 µm,Llama31-70B,	10 µm,Llama31-405B,	 1000 µm,o1-preview,LVError
Single LLM Baseline Run 4	 1 µm,GPT-4o,	1000 µm,Claude3-5Sonnet,	10 µm,Llama31-70B,	1000 µm,Llama31-405B,	 1000 µm,o1-preview,
Single LLM Baseline Run 5	 1000 µm,GPT-4o,	 10 µm,Claude3-5Sonnet,	10 µm,Llama31-70B,	 0.01 µm,Llama31-405B,	 1000 µm,o1-preview,

Table 6: Arrow Task Question: Generate an Arrow pointing to the right with length 10 mm, make the body 1/3 width of the head, start at 0,0.

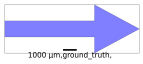
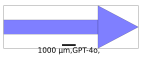
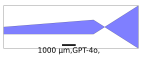
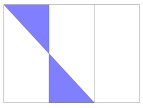
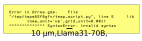
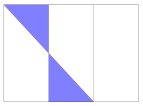
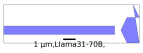
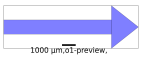
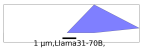
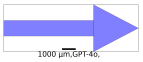
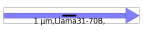
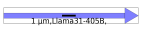
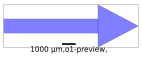
Ground Truth	GPT-4o	Claude-3.5	Llama-3-70B	Llama-3-405B	o1-preview
					
SOLOMON Single LLM Baseline Run 1	 		 		
Single LLM Baseline Run 2					
Single LLM Baseline Run 3					
Single LLM Baseline Run 4					
Single LLM Baseline Run 5					

Table 7: BasicLayout Task Question: 1. Draw a rectangular active region with dimensions 10 µm x 5 µm. 2. Place a polysilicon gate that crosses the active region vertically at its center, with a width of 1 µm. 3. Add two square contact holes, each 1 µm x 1 µm, positioned 1 µm away from the gate on either side along the active region.

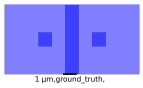
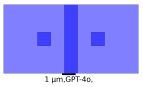
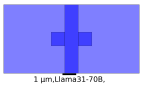
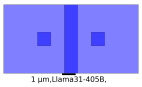
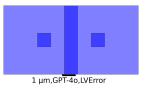
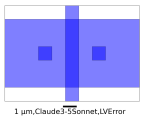
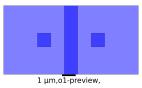
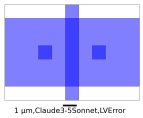
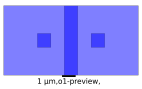
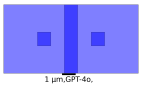
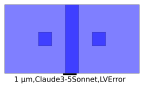
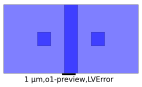
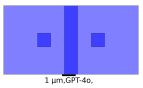
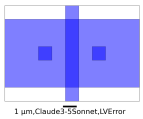
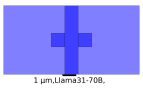
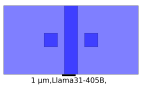
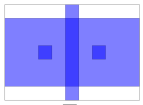
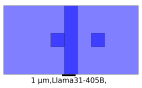
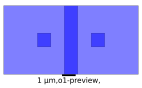
Ground Truth	GPT-4o	Claude-3.5	Llama-3-70B	Llama-3-405B	o1-preview
 1 µm,ground_truth,					
SOLOMON	 1 µm,GPT-4o,	1 µm,Claude3-5Sonnet,	 1 µm,Llama31-70B,	 1 µm,Llama31-405B,	
Single LLM Baseline Run 1	 1 µm,GPT-4o,LVError	 1 µm,Claude3-5Sonnet,LVError	10 µm,Llama31-70B,	1 µm,Llama31-405B,	 1 µm,o1-preview,
Single LLM Baseline Run 2	1 µm,GPT-4o,LVError	 1 µm,Claude3-5Sonnet,LVError	10 µm,Llama31-70B,	1 µm,Llama31-405B,	 1 µm,o1-preview,
Single LLM Baseline Run 3	 1 µm,GPT-4o,	 1 µm,Claude3-5Sonnet,LVError	10 µm,Llama31-70B,	1 µm,Llama31-405B,	 1 µm,o1-preview,LVError
Single LLM Baseline Run 4	 1 µm,GPT-4o,	 1 µm,Claude3-5Sonnet,LVError	 1 µm,Llama31-70B,	 1 µm,Llama31-405B,	1 µm,o1-preview,
Single LLM Baseline Run 5	0.1 µm,GPT-4o,	 1 µm,Claude3-5Sonnet,LVError	10 µm,Llama31-70B,	 1 µm,Llama31-405B,	 1 µm,o1-preview,

Table 8: ViaConnection Task Question: Create a design with three layers: via layer (yellow), metal layer (blue), and pad layer (red). The via radius is 10 units, pad radius is 30 units, and metal connection width is 40 units with a total length of 600 units. Position the first via at (50, 150) and the second via at (550, 150). Ensure the metal connection fully covers the vias and leaves a margin of 10 units between the edge of the metal and the pads. Leave a space of 50 units between the vias and the edges of the metal connection.

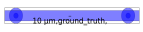
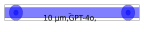
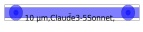
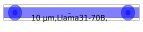
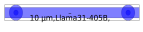
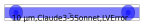
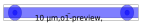
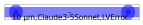
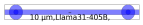
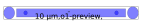
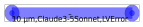
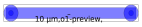
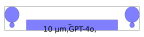
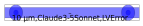
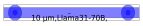
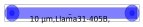
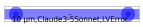
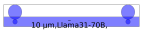
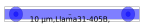
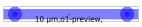
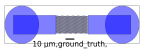
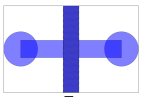
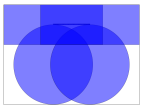
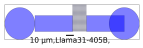
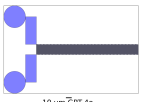
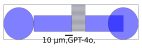
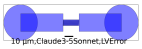
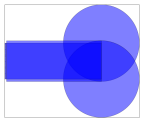
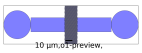
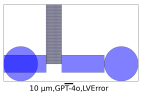
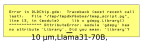
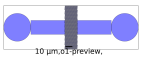
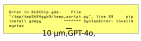
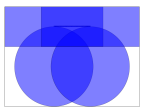
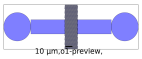
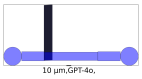
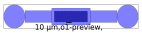
Ground Truth	GPT-4o	Claude-3.5	Llama-3-70B	Llama-3-405B	o1-preview
					
SOLOMON					
Single LLM Baseline Run 1					
Single LLM Baseline Run 2					
Single LLM Baseline Run 3					
Single LLM Baseline Run 4					
Single LLM Baseline Run 5					

Table 9: DLDChip Task Question: Draw a deterministic lateral displacement chip - include channel that can hold the array has gap size = 225 nm, circular pillar size = 400 nm, width = 30 pillars, row shift fraction = 0.1, add an inlet and outlet 40 µm diameter before and after the channel, use a 20*50 µm bus to connect the inlet and outlet to the channel.

Ground Truth	GPT-4o	Claude-3.5	Llama-3-70B	Llama-3-405B	o1-preview
 10 µm,ground_truth,					
SOLOMON	10 µm,GPT-4o,	 10 µm,Claude3-5Sonnet,	 1 µm,Llama31-70B,	 10 µm,Llama31-405B,	
Single LLM Baseline Run 1	 10 µm,GPT-4o,	10 µm,Claude3-5Sonnet,LVError	10 µm,Llama31-70B,	1 µm,Llama31-405B,	10 µm,o1-preview,
Single LLM Baseline Run 2	 10 µm,GPT-4o,	 10 µm,Claude3-5Sonnet,LVError	10 µm,Llama31-70B,	 1 µm,Llama31-405B,	 10 µm,o1-preview,
Single LLM Baseline Run 3	 10 µm,GPT-4o,LVError	10 µm,Claude3-5Sonnet,LVError	 10 µm,Llama31-70B,	10 µm,Llama31-405B,	 10 µm,o1-preview,
Single LLM Baseline Run 4	 10 µm,GPT-4o,	10 µm,Claude3-5Sonnet,LVError	10 µm,Llama31-70B,	 1 µm,Llama31-405B,	 10 µm,o1-preview,
Single LLM Baseline Run 5	 10 µm,GPT-4o,	10 µm,Claude3-5Sonnet,LVError	10 µm,Llama31-70B,	10 µm,Llama31-405B,	 10 µm,o1-preview,

Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction accurately reflect the paper's contributions and scope, including the introduction of SOLOMON architecture, evaluation methods, and limitations.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The paper discusses limitations in Section 5 (Conclusion and Future Work), addressing challenges in translating domain knowledge and areas for improvement.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory Assumptions and Proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [NA]

Justification: The paper does not include theoretical results requiring proofs.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental Result Reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The paper provides detailed information on the experimental setup, including task descriptions and evaluation methods, allowing for reproducibility.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: The complete benchmark dataset of 25 tasks and the code to call 5 LLMs are released under the Apache 2.0 license at github.com/wenboown/generative-ai-for-semiconductor-physical-design. This includes LLM outputs, extracted Python code, error logs, and converted PNGs for reproducibility. The informaiton is provided in Appendix A.1.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental Setting/Details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: Yes, the paper describes the experimental methodology and design philosophy in detail. The optimization of the system is done through prompt engineering. The prompts and evaluation metrics are provided in Appendix A. The evaluation code are open-sourced.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment Statistical Significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [NA]

   Justification: This is a proof-of-concept paper with a pilot study. The results are not statistically significant.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments Compute Resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The paper specifies that LLM experiments were conducted by calling APIs (GPT-4o via OpenAI, Claude via Anthropic, Llama 3s via IBM watsonx). Local code was run on a VM with RHEL 8.0, 32 core CPU, and 256GB memory. The information is provided in Appendix A.1.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code Of Ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research conform to the NeurIPS Code of Ethics, discussions of potential societal impacts and ethical considerations are brief but present in the Introduction and Conclusion and Future Work.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader Impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

Justification: The paper discusses both potential positive and negative societal impacts in the Conclusion section.

Guidelines:

- The answer NA means that there is no societal impact of the work performed.
- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

Answer: [NA]

Justification: The paper does not release high-risk models or datasets requiring specific safeguards.

Guidelines:

- The answer NA means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: The paper mentions using GPT-4o, Claude, and Llama 3s via their respective APIs. The benchmark dataset and code are released under the Apache 2.0 license.

Guidelines:

- The answer NA means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.

- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New Assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: The new benchmark dataset of 25 tasks and associated code are released with documentation in the github repository. While SOLOMON code is not included due to proprietary reasons, its output results are open-source and included in the repository. The information is provided in Appendix A.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and Research with Human Subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: The paper does not involve crowdsourcing or research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

Justification: The research does not involve human subjects requiring IRB approval.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.