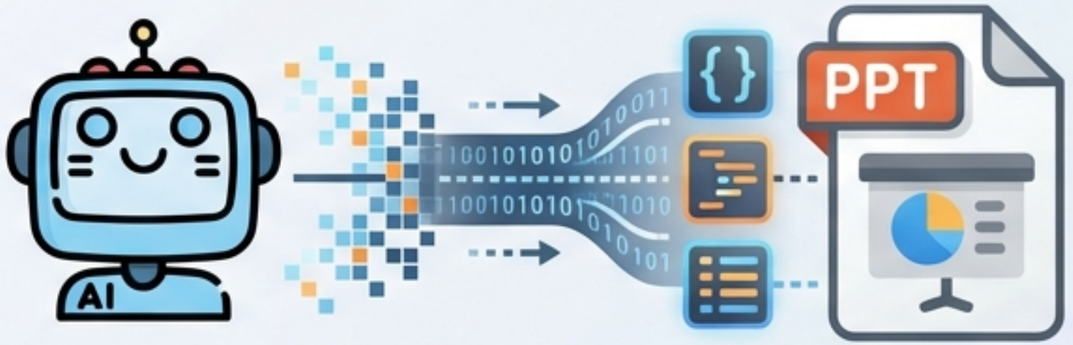
SlideCoder: Layout-Aware RAG-Enhanced Hierarchical Slide Generation

Converting Reference Images into Editable PowerPoint Scripts with Precision



Wenxin Tang, Jingyu Xiao, Wenxuan Jiang, Xi Xiao, Yuhang Wang, Xuxin Tang, Qing Li, Yuehe Ma, Junliang Liu, Shisong Tang, Michael R. Lyu.

Tsinghua University | CUHK | Northeastern University | Kuaishou Technology | Peng Cheng Laboratory

Context

The Challenge: Automated Slide Creation Fails at Fidelity

Current **Multimodal** LLMs (MLLMs) like AutoPresent attempt to automate slide creation from natural language. However, when tasked with replicating a visual **Reference Image (RI)**, they fail to capture the structural integrity of the design.

Visual Evidence

1. **Miss**: Complete omission of visual or textual elements.
2. **Incorrect**: Deviations in style (font weight, color, shape attributes).
3. **Disorder**: Significant errors in spatial arrangement and alignment.

Why MLLMs Struggle with Slide Generation

Visual Blindness

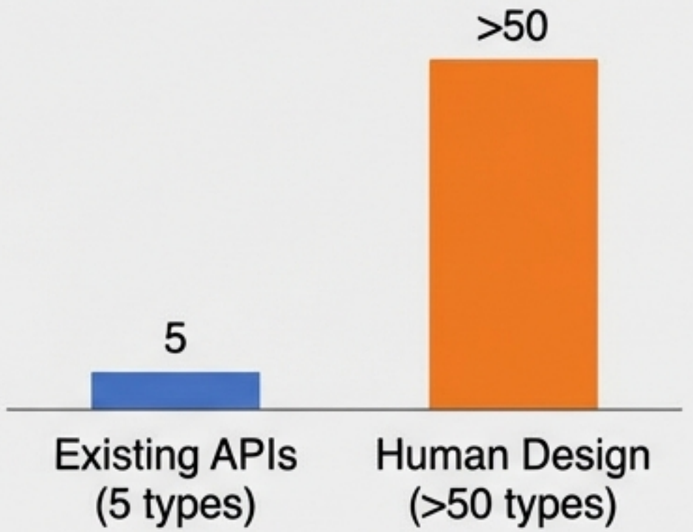
Natural language prompts cannot capture pixel-perfect layout nuances or the high density of elements in professional slides.

Library Ignorance

```
slide.shapes.add_magic_box()
```

General purpose MLLMs hallucinate python-pptx code, often using invalid syntax or non-existent APIs.

The Complexity Gap

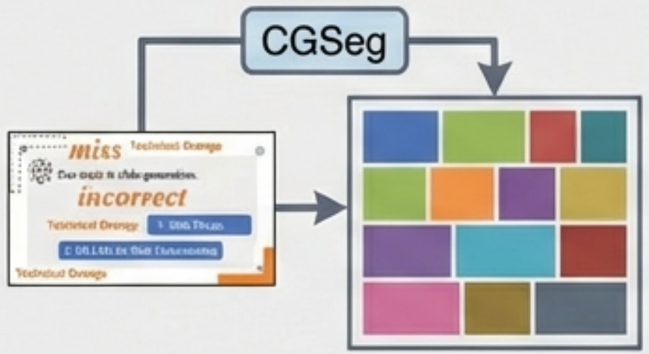


Current methods rely on simplified APIs (SLIDLILIB) that support only basic operations, failing to capture complex human design intentions.

Introducing SlideCoder: A Unified End-to-End Framework

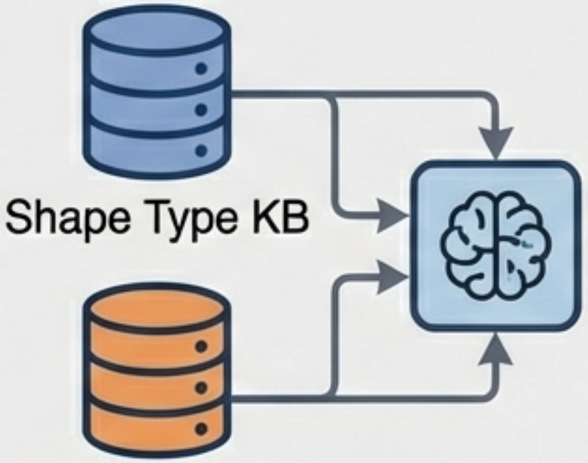
Divide and Conquer via Hierarchical Generation

Segmentation



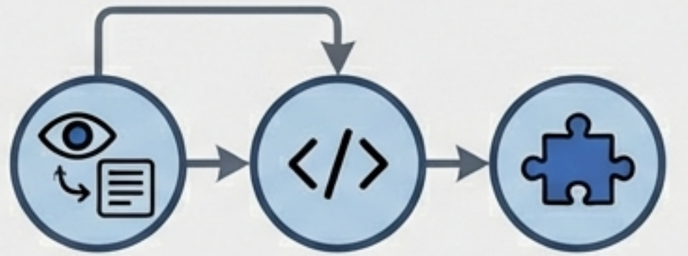
Decomposes slide images into semantically meaningful regions.

Hierarchical RAG



Injects specific knowledge to prevent code hallucinations.

Agent Workflow



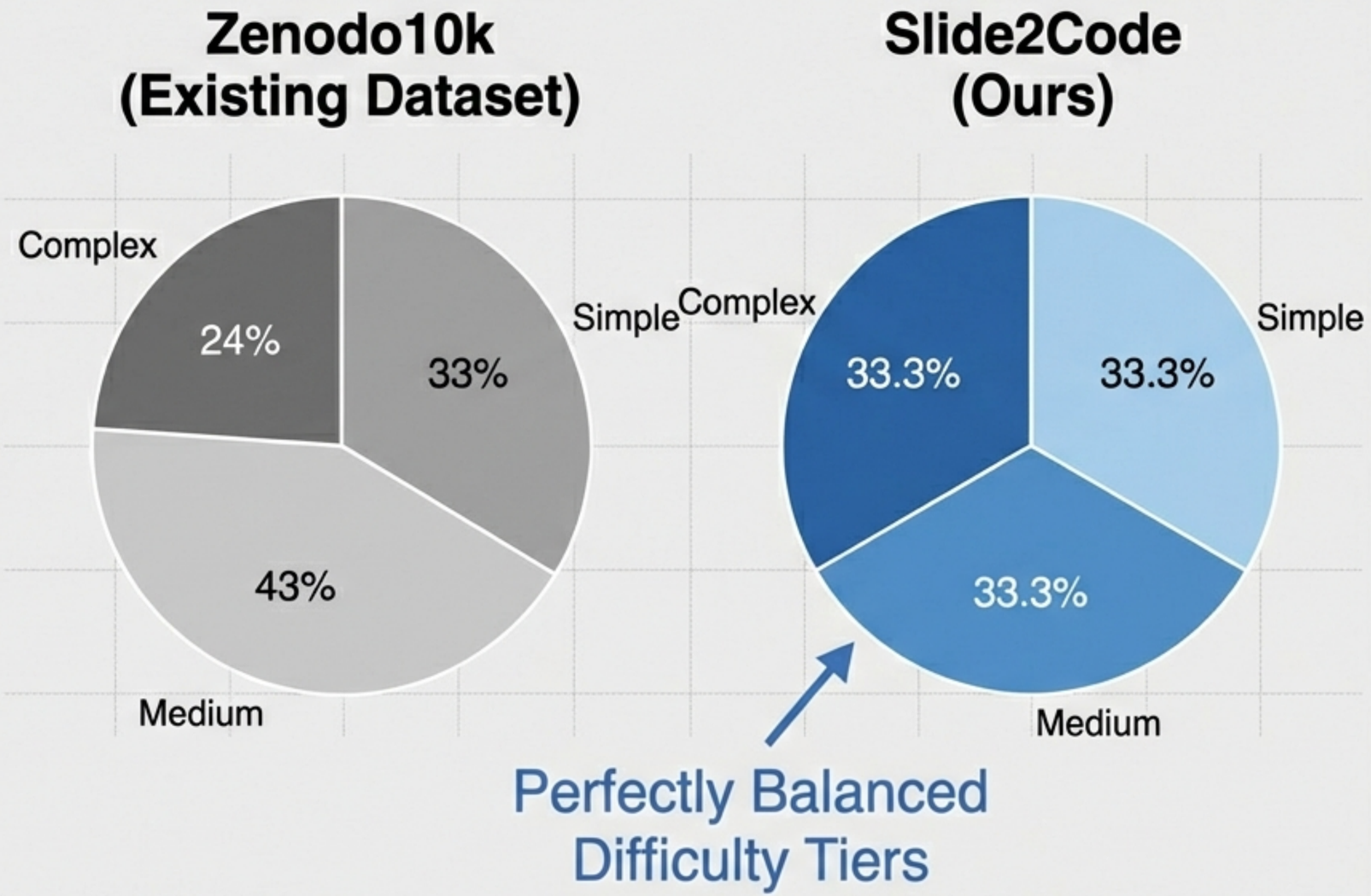
Multi-agent relay to generate and assemble final code.

Contribution 1: The Slide2Code Benchmark

New Task:
Reference Image (RI) to Slide Generation.

Goal: Generate Python code that replicates a provided image.

Data Source: Curated from ~32,000 candidates to select 300 high-quality samples.



Measuring Difficulty: The Slide Complexity Metric (SCM)

$$z_i = \alpha \cdot \tilde{c}_i + \beta \cdot \tilde{e}_i + \gamma \cdot \tilde{v}_i$$

Variable: c_i

Element Count

Total number of visual elements on the canvas.

Variable: e_i

Element Diversity

Count of distinct element types (textboxes, charts, shapes).

Variable: v_i

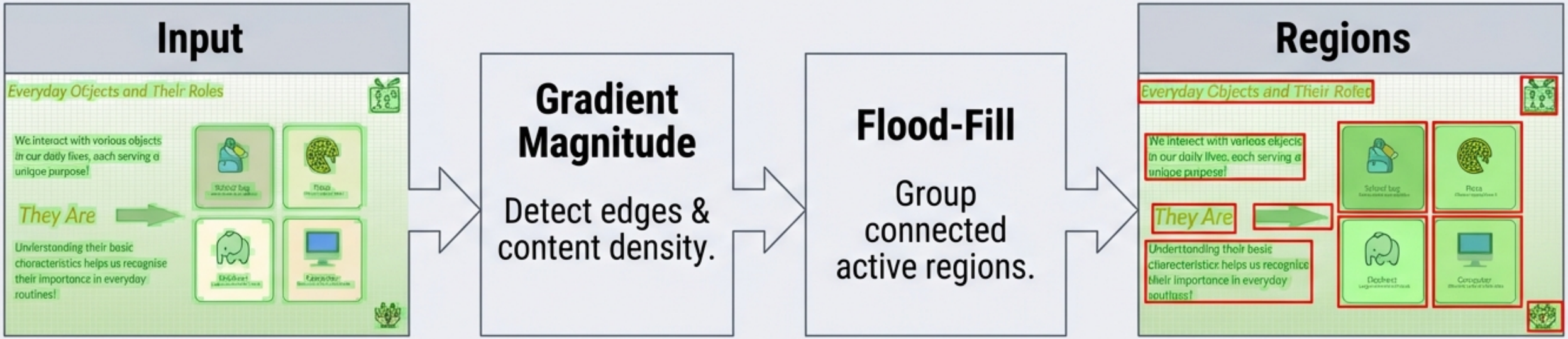
Visual Density

Element Coverage Ratio calculated via activated color grids.

Validation: SCM correlates strongly with human judgment ($r = 0.873$).

Methodology I: Color Gradient-based Segmentation (CGSeg)

Recursive decomposition based on visual density.



Methodology II: Hierarchical RAG (H-RAG)

Dual-layer knowledge injection for syntax correctness.

Concept

Level 1: Shape Type KB

Target Agent: Describer
Function: Standardizes terminology (e.g., 'Placeholder' vs 'Box').

```
{'Auto Shape': 'Predefined shape with approx 180 variations...'}
```

Level 2: Operation Function KB

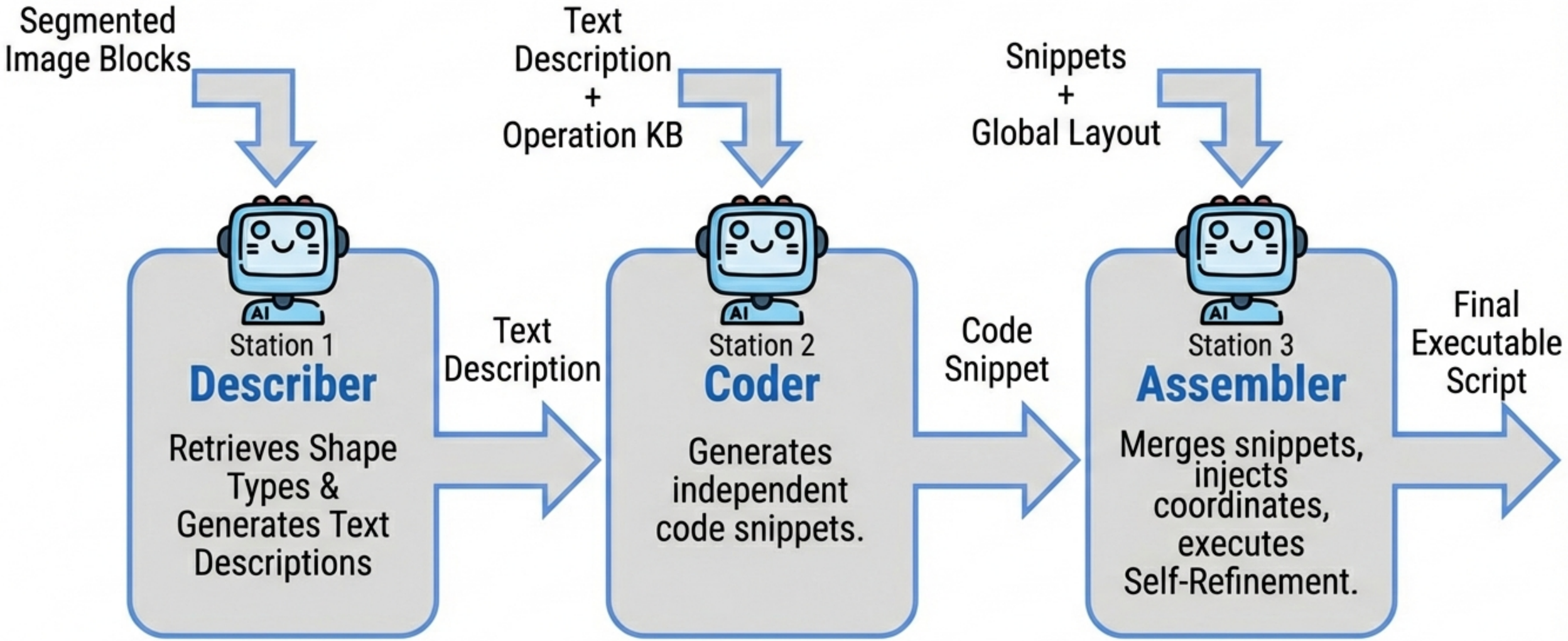
Target Agent: Coder
Function: Provides exact API syntax, parameters, and return values.

```
pptx.Presentation(pptx=None) -> Presentation object
```

Implementation

You are a python-pptx expert...
The following is an introducti to layout shape shape types in PPT:
{<Grammar>}

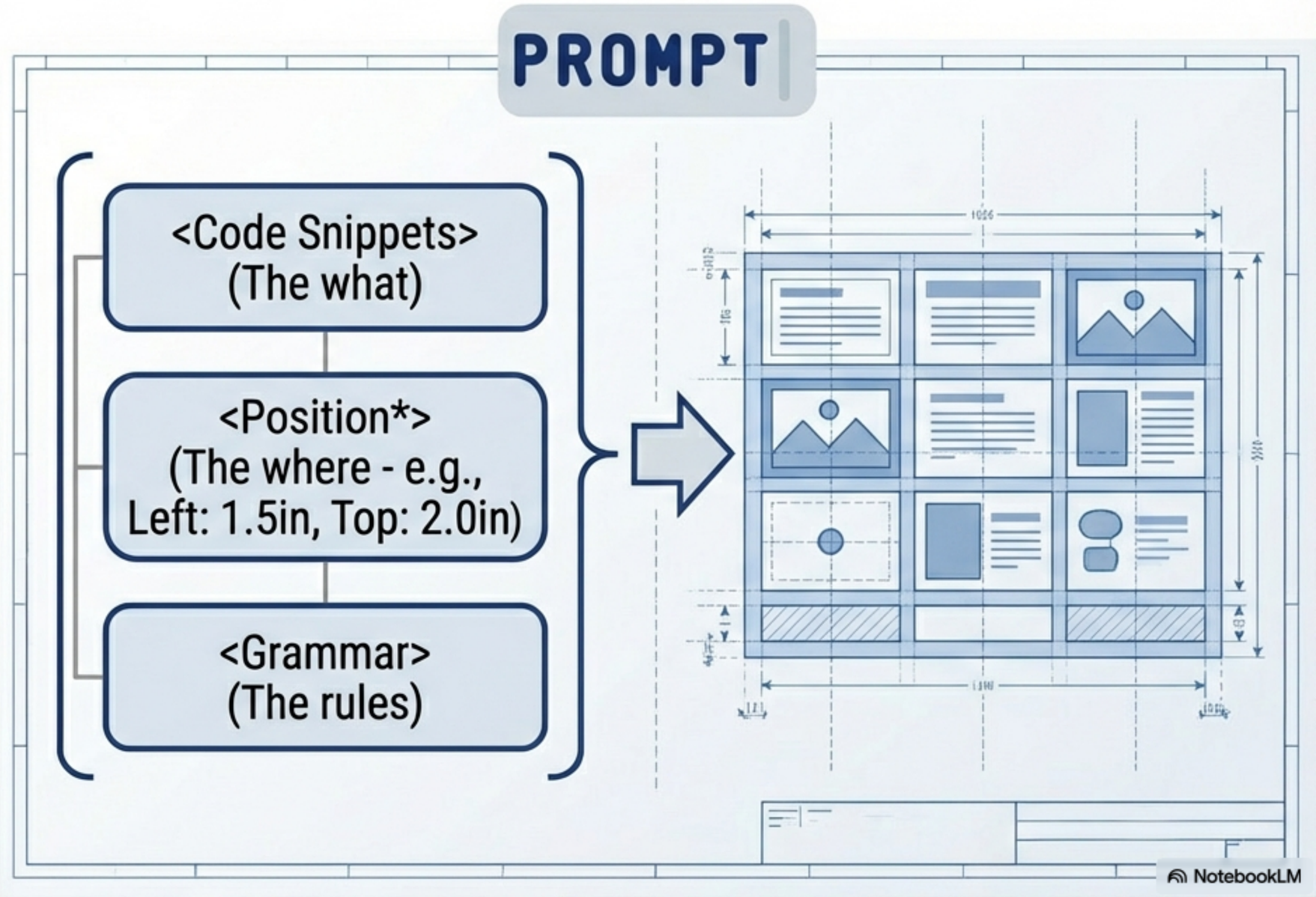
Methodology III: The Multi-Agent Workflow



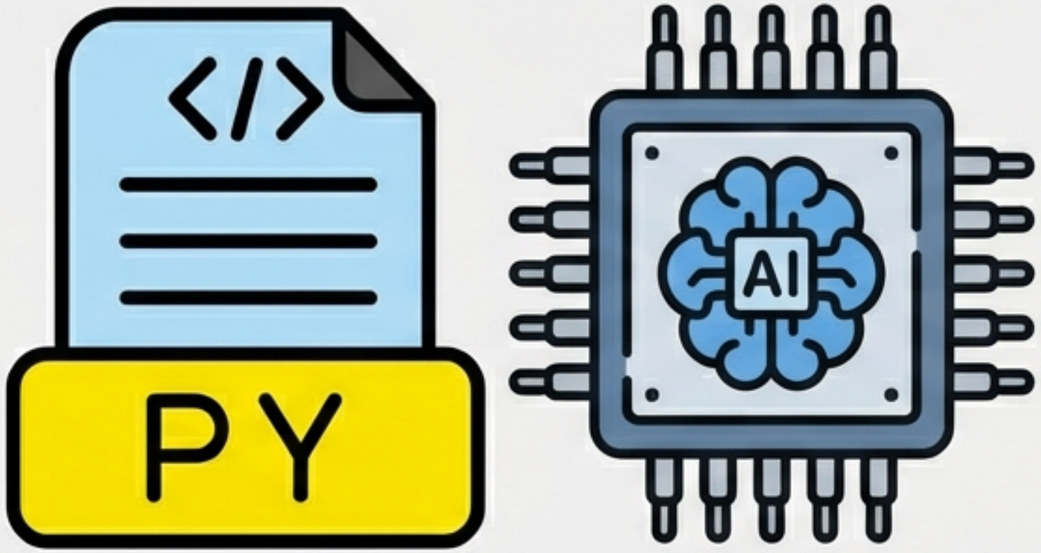
Methodology IV: Layout-Aware Prompting

The Challenge:
Local code snippets don't know where they belong on the global canvas.

The Fix:
Injecting relative coordinates (x, y, w, h) derived from CGSeg directly into the prompt.



SlideMaster: The Open-Source Contender



Base Model: Qwen2.5-VL-7B-Instruct

Training: Fine-tuned on high-quality (Image, Code) pairs.

The Secret Sauce: Custom PPTX Reverse-Engineering Tool

Feature	AutoPresent (Baseline)	SlideMaster (Ours)
Object Types	5	**10**
Styles	16	**44**
Complex Support	Basic Shapes	Tables, Connectors, Gradients

Quantitative Results: Dominating the Leaderboard

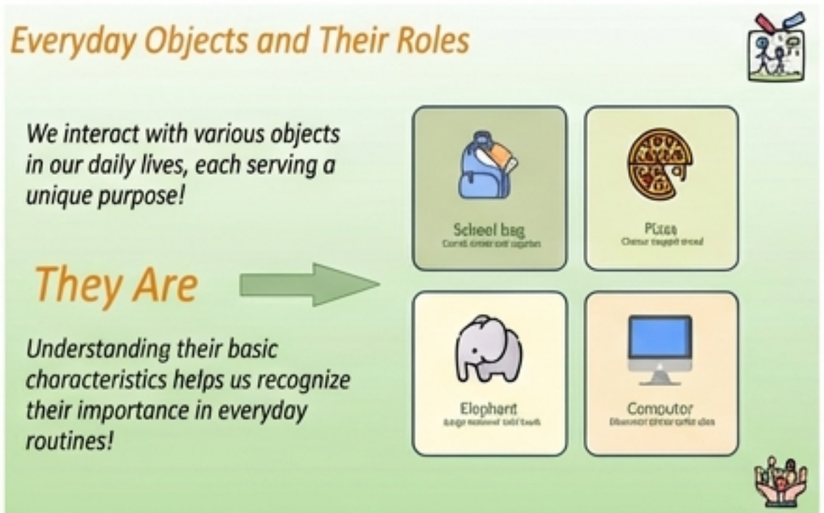
Framework	Difficulty	Execution Rate	Overall Score
AutoPresent (GPT-4o)	Simple	61.0%	80.3
Technical Green	Complex	41.4%	**53.0**
SlideCoder (GPT-4o)	Simple	**99.0%**	**91.8**
Technical Green	Complex	**85.5%**	**82.2**

On Complex tasks, SlideCoder outperforms the baseline by **40.5 points**.

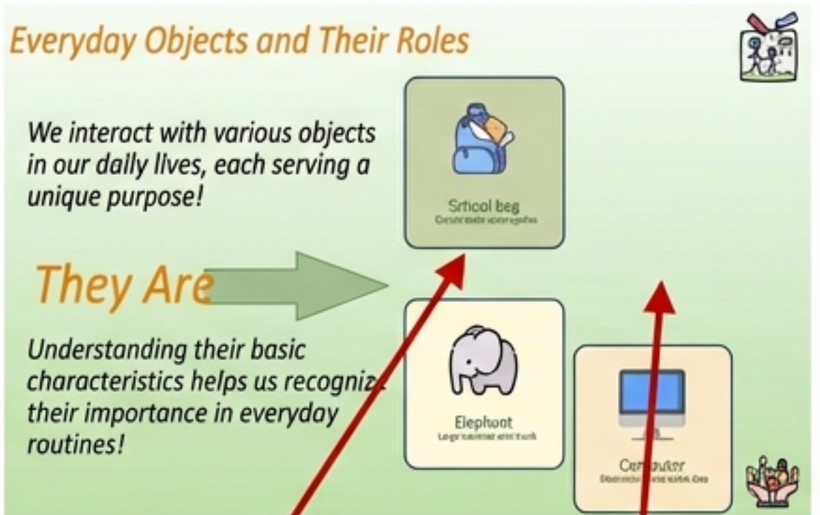
Execution Rate approaches **100%**, compared to ~40-60% for baselines.

Qualitative Comparison: Visual Fidelity

Reference Image (Input)



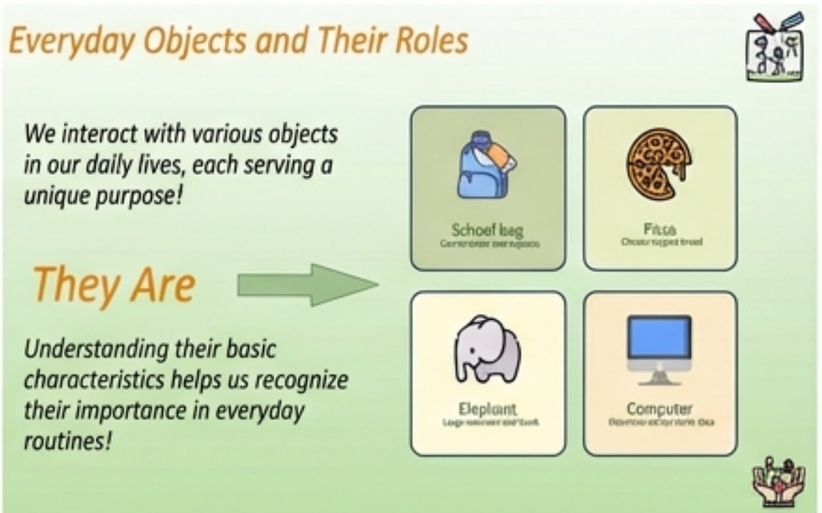
Baseline (AutoPresent)



Disorder

Miss

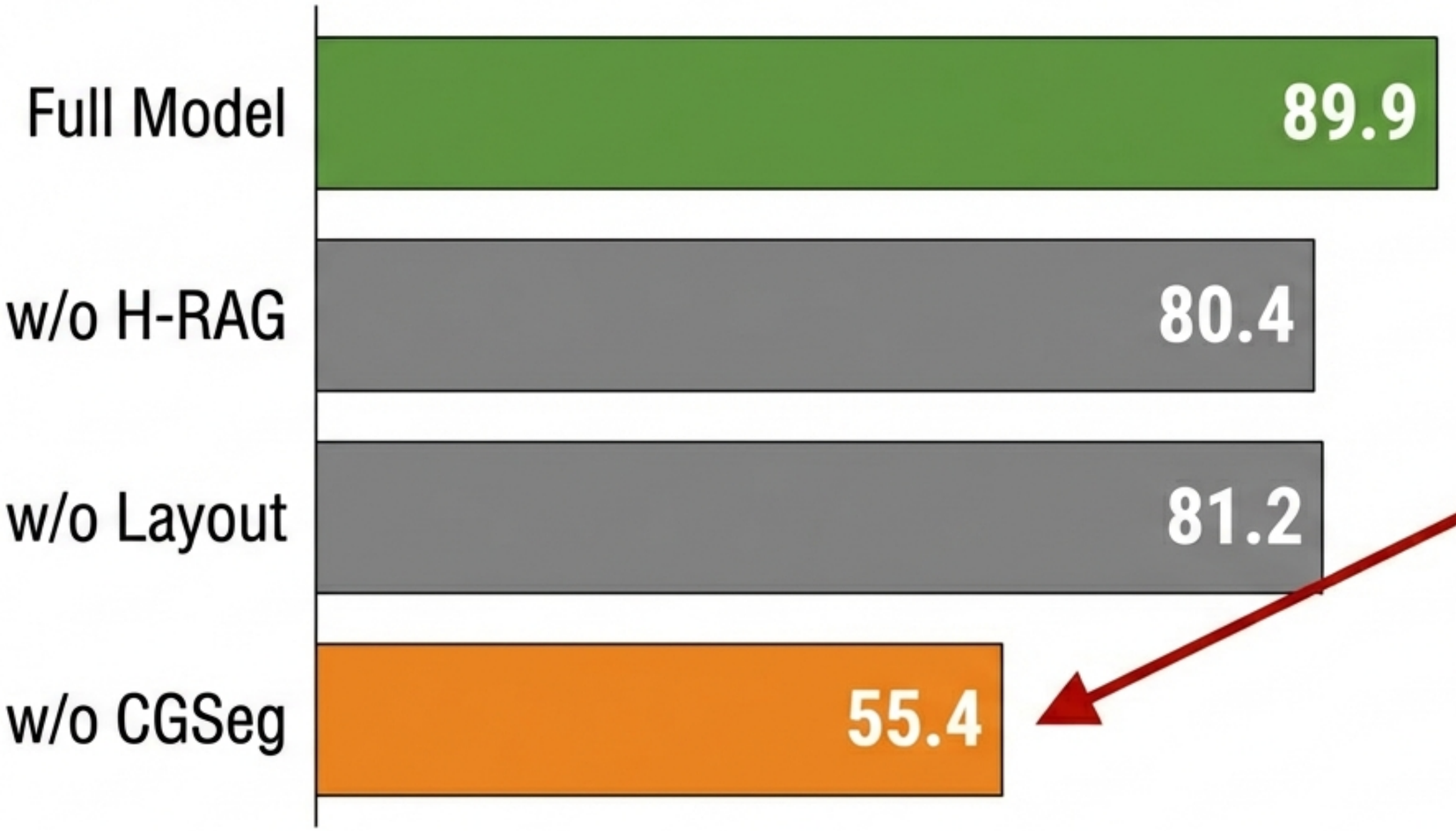
SlideCoder (Ours)



Preserves grid layout, icon placement, and text hierarchy.

Ablation Study: Why Every Component Matters

Impact of Removing Components (Overall Score on Simple Tasks)



Removing Segmentation (CGSeg) causes a massive performance drop. The model gets overwhelmed by the full slide image.

Conclusion & Future Directions

- **Slide2Code**: Established a rigorous new benchmark for complex slide generation.
- **SlideCoder**: Achieved SOTA by integrating Segmentation (CGSeg), Retrieval (H-RAG), and Layout-Aware Prompting.
- **SlideMaster**: Released an open-source 7B model that rivals proprietary giants like GPT-4o.

Future Work:

- Multi-slide generation capabilities.
- Handling full-slide background images without separate assets.

Code & Data Available: https://github.com/vinsontang1/SlideCoder