Supplementary Material: AI-Generated Lecture Slides for Improving Slide Element Detection and Retrieval

No Author Given

No Institute Given

The goal of this appendix is to provide an in-depth description of the **SynLecSlideGen** Pipeline and provide visual results for both the experimented tasks. A detailed overview of the pipeline can be obtained from Figure 1

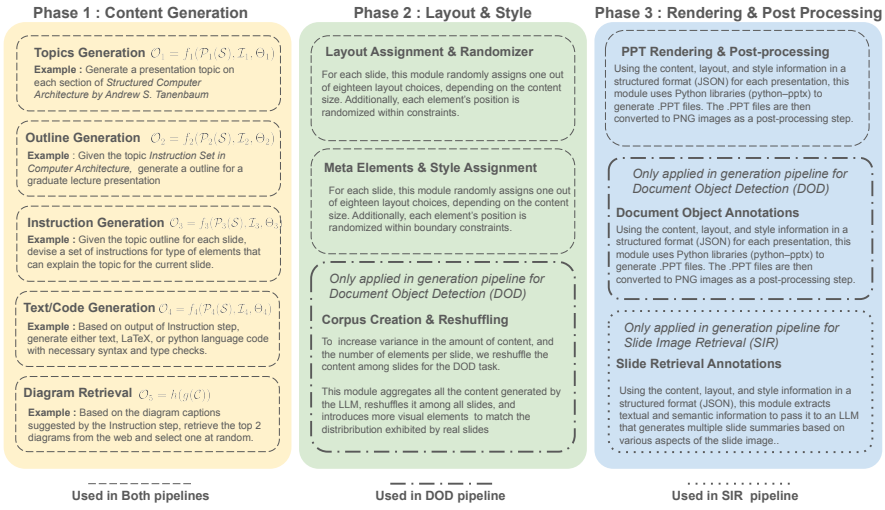


Fig. 1: Detailed overview of each module in the **SynLecSlideGen** pipeline.

A Detailed Prompting Techniques used for Content Generation

A.1 Model Configurations and Settings

Table 1 summarizes the LLM configurations used across the four stages of the generation pipeline. We utilized state-of-the-art models such as GPT-4 and GPT-3.5, selected for their ability to handle complex, multi-modal content generation tasks.

Stage	**Model**	**Temperature** (T)	**Top-p** (p)	**Max Tokens** (L)
Topic Generation	GPT-4	0.7	0.9	256
Outline Generation	GPT-4	0.6	0.95	512
Instruction Generation	GPT-3.5	0.7	0.9	1024
Text/Code Generation	GPT-3.5	0.5	0.8	2048

Table 1: Model configurations and hyperparameter settings for each stage.

A.2 Detailed Prompting Techniques

We provide examples of prompts, instructions, and outputs for each stage of the pipeline to illustrate our approach. The outputs demonstrate the progression from initial topic generation to fully realized slide content.

Topic Generation In this stage, the model generates a list of potential topics based on a seed input, such as a book title or a course name.

Prompt Provided:

```
System Prompt:
You are a helpful assistant to a professor.
You have access to all resources on the web.

User Prompt:
I am a {subject} professor and I want to create
lecture slides for various topics in the book
: {book} by {author}.
Give me a list of topics from the book I provided
in the form of a Python dict.
Also, I will use these topics to make presentations
and hence augment the presentation titles if they are generic
For example: for titles like Introduction, Applications
convert them to Introduction to Deep Learning,
Applications of Deep Learning, respectively.
Do not provide any converstation.
```

One Shot Example:

```
Here's an example

Book Name : Understanding Deep Learning
Subject : CS
Expected Output:
{{
    "CS": [
        "Math for Deep Learning Basics",
```

```
"Intro to Supervised Learning",
"Shallow Neural Networks",
"Activation Functions",
"Composing Neural Networks",
"Gradient Descent Optimization",
"Stochastic Gradient Descent",
"Adam Optimization Algorithm",
"Backpropagation in Toy Model",
"Initialization Techniques",
"MNIST-1D Performance Analysis",
"Bias-Variance Trade-off",
"Double Descent Phenomenon",
"L2 Regularization Techniques",
"Implicit Regularization Methods",
"Model Ensembling Techniques",
"Bayesian Methods in ML",
"Data Augmentation Techniques",
"1D Convolution Basics",
"Convolution for MNIST-1D",
"2D Convolution Basics",
"Downsampling & Upsampling",
"Shattered Gradients Issue",
"Residual Networks Introduction",
"Batch Normalization Role",
"Self-Attention Mechanisms",
"Multi-Head Self-Attention",
"Graph Encoding Techniques",
"Graph Classification Methods",
"Neighborhood Sampling",
"Graph Attention Mechanisms",
"GAN Toy Example",
"Wasserstein Distance in GANs",
"1D Normalizing Flows Intro",
"Autoregressive Flows Intro",
"Latent Variable Models Intro",
"Reparameterization Trick",
"Importance Sampling Methods",
"Diffusion Encoder Basics",
"1D Diffusion Model Basics",
"Reparameterized Model Intro",
"Diffusion Models Families",
"Markov Decision Processes Intro",
"Dynamic Programming Basics",
"Monte Carlo Methods Intro",
"Temporal Difference Methods",
```

```
        "Control Variates Methods",
        "Random Data Generation",
        "Full-Batch Gradient Descent",
        "Lottery Tickets Hypothesis",
        "Adversarial Attacks Techniques",
        "Bias Mitigation Strategies",
        "Explainability Techniques"
    ]
}}
""")]
```

Settings: GPT-4, $T = 0.7$, $p = 0.9$, $L = 256$.

Outline Generation The topics generated in the previous stage are expanded into a detailed outline.

Prompt Provided:

```
System Prompt:
You are a helpful university professor and you
are guiding your PhD student to create an outline
for the lecture he will deliver to a class.



User Prompt:
I would like to get help designing a detailed
Table of Contents for an advanced university
presentation lecture on {topic} based on the book {book}.
Please help me create the Table of Content in form of a Python dict.
```

One Shot Example:

```
Example:
Input: Tree Data Structures based on the book
Data Structures and Algorithms made easy.

Expected Output:
[
"Introduction : Definition & Characteristics",
"Introduction : Example of a Tree",
"Types of Trees",
"Binary Trees : What are they?",
"Binary Trees : Searching an element",
```

```
"Tree Traversal",
"Pre-order Traversal",
"In-order Traversal",
"Post-order Traversal",
"Comparing traversal methods",
"Binary Search Trees: Introduction",
"Binary Search Trees: Time Complexity",
"BST v/s Binary Trees",
"Applications of Trees",
"Huffman Algorithm : History",
"Huffman Algorithm : Pseudocode",
"Summary of Trees"
]

I want you to provide the output in form of a python
list of strings having slide titles of each slide.
The length of the list will be the total number of slides
in the presentation. Do not generate more than 15 slide
titles and each title should have maximum of 5 words.
Just return the output without the conversation.""")
]
```

Settings: GPT-4, $T = 0.6$, $p = 0.95$, $L = 512$.

Instruction Generation This stage translates the outline into detailed instructions for creating slide content.

Prompt Provided:

```
User Prompt:

Hello. I want you to help me prepare lecture slides on {topic}.
I am providing you with the outline of the lecture
in form of a nested dictionary.

Outline :{outline}
Each element in the object is a slide where the value
represent the slide title. I want you to add two keys
namely 'element_type' and 'element_caption' for each
subsection and determine which types of elements would
be most beneficial to understand that subsection.

The elements can be as follows: {elements}.
You should generate exactly two elements per slide.
Do not generate more of less elements per slide.
Whenever possible generate atleast one text based element
```

```
(Description, URL, or Enumeration) and one visual element
(Rest of the elements) per subsection, such that there
is diversity in elements.

As a rule of thumb, make sure the distribution of elements
is nearly same for the entire presentation.
I want you to generate the results within the outline
and only output the revised outline without any conversation.
Do not generate the slide numbers in the output,
they are just for your reference.
Your output should be in form of a Python Dict.
```

Expected Output:

```
{
    "Introduction to Gaussian Distributions": [
        {"element_type": "description", "element_caption":
        "Overview of Gaussian distributions and their
        importance in statistics"},
        {"element_type": "graph", "element_caption": "Visual
        introduction to the bell curve shape of Gaussian
        distributions"}
    ],

    "Historical Background": [
        {"element_type": "description", "element_caption":
        "Discussion on the origin and development of Gaussian
        distributions"},
    ],
    "The Normal Distribution: Definition": [
        {"element_type": "description", "element_caption":
        "Formal definition of the normal distribution"},
        {"element_type": "equation", "element_caption":
        "Mathematical equation of the normal distribution"}
    ],
    "Properties of Gaussian Distributions": [
        {"element_type": "block-diagram", "element_caption":
        "Diagram showing key properties such as symmetry and
        bell shape"},
        {"element_type": "enumeration", "element_caption":
        "List of statistical properties like mean, variance,
        etc."}
    ],
    ...
}
```

Settings: GPT-3.5, $T = 0.7$, $p = 0.9$, $L = 1024$.

Text/Code Generation In the final stage of synthetic data generation, we classify elements based on their type to construct specialized prompts for generating content. Each class type corresponds to a distinct content generation mode:

- **Textual elements:** Captions, headings, descriptions, and enumerations, for which simple text is generated.
- **LaTeX-rendered elements:** Tables and equations, for which LaTeX code is generated.
- **Code-rendered elements:** Plots, charts, and diagrams, for which code in Python (Matplotlib) or DOT language is generated.

This approach ensures that the model adheres to specific output requirements for different types of slide content. Below, we describe the process using pseudocode and provide example prompts for a textual "description" element and a LaTeX-rendered "equation" element.

Pseudocode for Prompt Construction The construction of prompts involves iterating through each slide's elements, determining the type of each element, and appending specialized instructions to the respective prompt. The pseudocode for this process is as follows:

```
1. Initialize three prompts:
   a) Text prompt for textual elements.
   b) Structural prompt for LaTeX-rendered elements.
   c) Code prompt for plots and diagrams.

2. For each slide:
   a) Iterate through all elements in the slide.
   b) Identify the element type
   (e.g., "description", "equation", "plot").
   c) Generate type-specific instructions:
      i. For "textual" types:
         Append context and instructions for generating simple text.
      ii. For "LaTeX-rendered" types:
         Append context and instructions for generating LaTeX code.
      iii. For "code-rendered" types:
         Append context and instructions for generating Matplotlib
         or DOT code.

3. Add post-generation checks to each prompt:
   a) Ensure syntax correctness.
   b) Verify that the number of generated snippets matches the requests.
```

Example Prompts

Prompt for a Textual Element (Description) This prompt instructs the model to generate descriptive text based on the provided caption and context.

```
For the section title "Overview of Machine Learning" (Slide Number 1),
generate a short descriptive text for the slide.
Caption: "Introduction to the basic concepts and applications of
Machine Learning."
```

Generated Output:

- Machine Learning is a field of Artificial Intelligence that focuses on using data-driven approaches to make predictions or decisions. Applications include image recognition, natural language processing, and recommendation systems.

Prompt for a LaTeX-Rendered Element (Equation) This prompt instructs the model to generate LaTeX code for an equation.

```
For the section title "Mathematics of Machine Learning" (Slide Number 2),
generate LaTeX code for a simple equation given the caption:
"Representation of a linear regression model."

Generate LaTeX code as plain text separated by ‘‘‘latex<content>‘‘‘.
Do not include equation numbers.
```

Generated Output:

```
‘‘‘latex
y = w_1x_1 + w_2x_2 + \dots + w_nx_n + b
‘‘‘
```

Post-Processing and Validation After generating all prompts and receiving the outputs, post-processing involves:

- **Syntax Verification:** Ensuring that the LaTeX and code snippets are syntactically correct.
- **Snippet Count Matching:** Confirming that the number of generated snippets matches the number of requests for each class type.

This modular approach to prompt construction allows for efficient generation of diverse content types while maintaining high output quality.

A.3 Summary

The multi-stage prompting strategy ensures semantic alignment and logical flow in slide content generation. By carefully designing prompts and tuning model settings, we successfully generated a dataset that mimics real-world lecture slide structures while remaining fully annotated for downstream tasks.

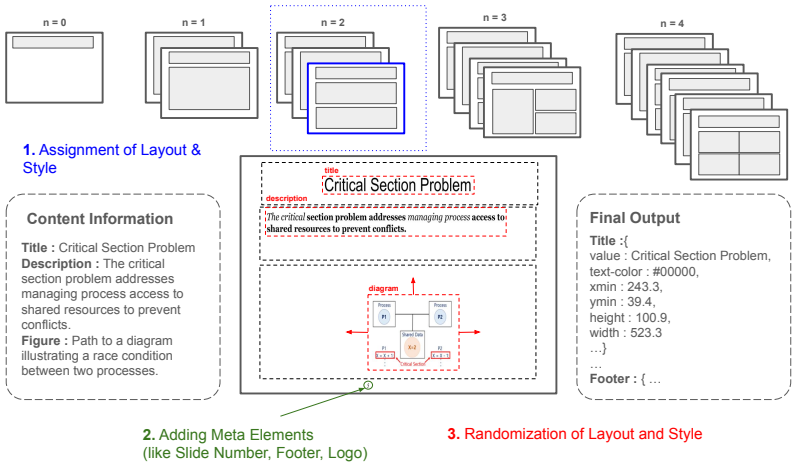


Fig. 2: Overview of Layout and Style Discriminator module.

B Layout and Style Discriminator Module

After generating content as textual descriptions and code snippets in Phase 1, the Layout and Style Discriminator Module is responsible for determining how this content is rendered into slides. This process involves selecting appropriate layouts based on the number of elements, introducing randomness for aesthetic variation, and assigning styles for consistent presentation. 2 provides a schematic overview of the process.

B.1 Layout Assignment Process

The layout assignment process ensures that the arrangement of elements on the slide follows logical and visually appealing structures. The process is as follows:

Defining Layout Templates Layouts are predefined using a Python class, where each layout specifies the dimensions and positions of the slide elements (e.g., title, body, footer). These layouts are parameterized using a scaling factor MUL_FAC to accommodate different resolutions.

Pseudocode: Layout Definitions

```
1. Initialize a dictionary ‘dimensions‘ that stores layouts for:
   a) Title: Single large text area at the top of the slide.
   b) Footer: Three equally spaced text areas at the bottom of the slide.
   c) Body: Configurations for 1 to 3 elements arranged in:
      i. Single column.
```

```
      ii. Two-column format.
      iii. Two-row format.
      iv. Grid layout (e.g., 3 columns).

2. Define a method ‘get_layout_dimensions(layout_id)‘:
   a) Match ‘layout_id‘ with predefined layouts.
   b) Update the ‘dimensions‘ dictionary with corresponding element positions.
   c) Return updated dimensions for rendering.
```

Random Layout Selection Layouts are assigned based on the number of content elements to be displayed in the slide. Each slide's content is classified by the number of elements, and a random layout is selected from permissible layouts for that count.

Pseudocode: Random Layout Selection

```
1. Create a mapping ‘layout_mapping‘:
   a) Keys: Number of elements on the slide.
   b) Values: List of permissible layout IDs.

2. Function ‘generate_random_layout(total_body_elements)‘:
   a) Input: Total number of elements.
   b) Output: Randomly choose a layout ID from ‘layout_mapping‘.
```

Position Perturbation for Randomness Given a slide element with dimensions:

- d_{left}: Left coordinate.
- d_{top}: Top coordinate.
- W: Width.
- H: Height.

We perturb the position and size of the element to introduce randomness while maintaining structural coherence. Let σ represent the standard deviation of the Gaussian noise added to each coordinate. The perturbation is calculated as follows:

Let the perturbed top and left coordinates be d'_{top} and d'_{left}, respectively.

$$d'_{\text{top}} = d_{\text{top}} + \frac{H - h_0}{2} + \text{clip}\left(\mathcal{N}(0, \sigma), -\frac{H - h_0}{2}, \frac{H - h_0}{2}\right) \tag{1}$$

$$d'_{\text{left}} = d_{\text{left}} + \frac{W - w_0}{2} + \text{clip}\left(\mathcal{N}(0, \sigma), -\frac{W - w_0}{2}, \frac{W - w_0}{2}\right) \tag{2}$$

Here:

- h_0: Adjusted height, scaled based on the element type.

- w_0: Adjusted width, scaled based on the element type.
- $\mathcal{N}(0, \sigma)$: Gaussian noise with mean 0 and standard deviation σ.
- $\text{clip}(x, a, b)$: A function that restricts x to the range $[a, b]$:

$$\text{clip}(x, a, b) = \begin{cases} a & \text{if } x < a \\ x & \text{if } a \leq x \leq b \\ b & \text{if } x > b \end{cases}$$

The adjusted height h_0 and width w_0 are computed as:

$$h_0 = \alpha \cdot H \tag{3}$$
$$w_0 = \alpha \cdot W \tag{4}$$

where α is a scaling factor sampled uniformly:

$$\alpha \sim \mathcal{U}(\tau, 1)$$

Here, τ is a predefined lower bound for the scaling factor.

The standard deviation σ for the Gaussian noise and the scaling factor α may vary depending on the type of element:

- **Title**:

$$\begin{aligned} \sigma &= 0.5 \\ h_0 &= 0.8 \cdot H \\ w_0 &= 0.8 \cdot W \end{aligned}$$

- **Body**:

$$\begin{aligned} \sigma &= 1.0 \\ h_0 &= \alpha \cdot H \quad \text{with } \alpha \sim \mathcal{U}(\tau, 1) \\ w_0 &= \alpha \cdot W \quad \text{with } \alpha \sim \mathcal{U}(\tau, 1) \end{aligned}$$

- **Footer**:

$$\begin{aligned} \sigma &= 0.2 \\ h_0 &= H \\ w_0 &= 0.8 \cdot W \end{aligned}$$

Pseudocode: Perturbing Layout Positions

```
1. Function ‘randomize_location(dims, element)‘:
   a) Inputs:
      i. ‘dims‘: Dictionary with position and size (top, left, height, width).
      ii. ‘element‘: Type of slide element (title, body, footer).
   b) Process:
      i. Adjust dimensions using scaling factors for height and width.
      ii. Perturb top and left coordinates using Gaussian noise (STD).
   c) Outputs: Updated (left, top, width, height) coordinates.
```

Handling Meta-Elements Meta-elements such as slide numbers, footers, dates, and presenter names are added at predefined positions (e.g., the bottom of the slide). Their positions are slightly randomized using the perturbation function.

B.2 Style Assignment

Styles determine the visual presentation of elements, including fonts, colors, and backgrounds. Randomized styles are assigned to ensure slides appear distinct yet coherent.

Style Assignment Steps:

1. Define a pool of permissible styles for:
   - Titles: Font size, color, alignment.
   - Body text: Font type, spacing, indentation.
   - Background: Solid colors, gradients, or images.
2. Randomly select styles for each slide element.
3. Apply styles consistently across all elements of the same type within a slide.

B.3 Example Workflow

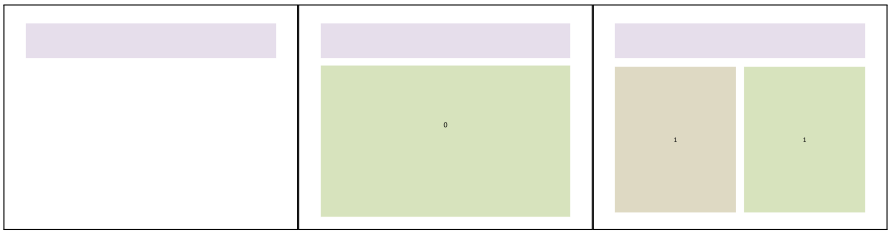
Consider a slide with two body elements. The workflow is as follows:

1. **Content Analysis:** Count the number of elements ($n = 2$).
2. **Random Layout Selection:** Choose a layout ID (e.g., ID=2) from permissible layouts for $n = 2$.
3. **Randomization:** Apply position perturbations to layout elements.
4. **Style Assignment:** Randomly assign styles to the title, body, and background.

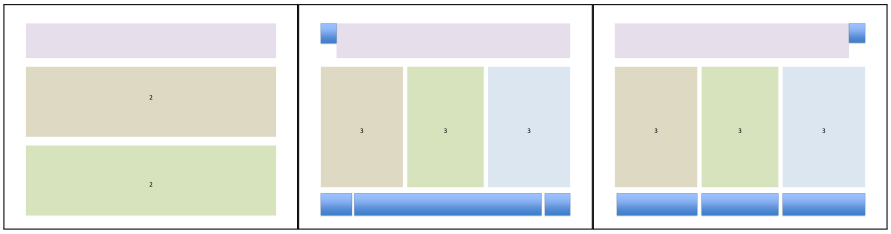
B.4 Type of Layouts and Design Templates

To sum up, the Layout and Style discriminator module achieves the following:

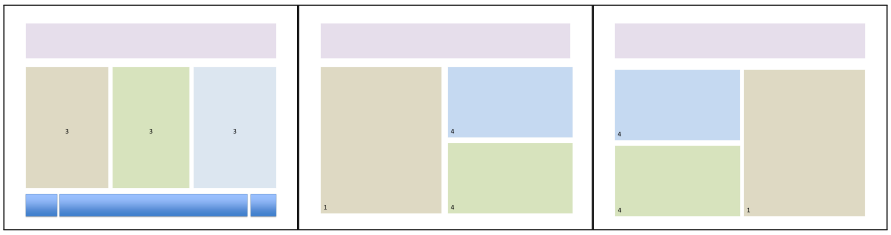
- Provides flexibility in slide design by using randomized layouts and styles.
- Ensures visual variety while maintaining structural coherence.
- Simplifies the transition from raw content to visually rendered presentations.



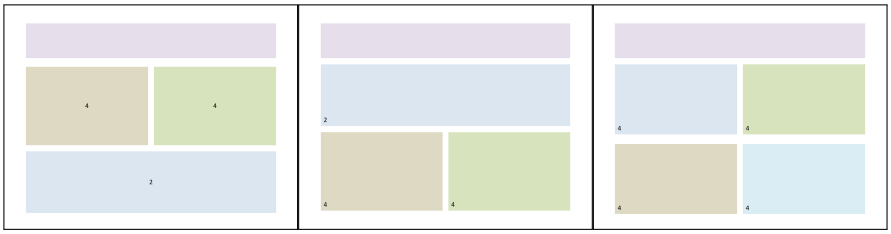
(a) n = 0 (b) n = 1 (c) n = 2



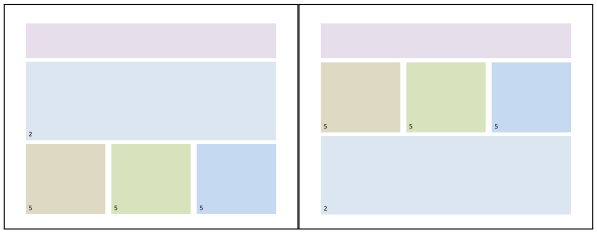
(d) n = 2 (e) n = 3 (f) n = 3



(g) n = 3 (h) n = 3 (i) n = 3



(j) n = 3 (k) n = 3 (l) n = 4



(m) n = 4 (n) n = 4

Fig. 3: Base Layouts used for SynSlide Generation

C Limitations of SynSlideGen

In this section, we discuss in detail the known limitations related to the **SynSlideGen** pipeline and the resulting **SynSlides** generated slides. Moreover, we provide data scaling results of **SynSlides** and its effect on downstream performance.

C.1 Effect of static layouts

As mentioned in Phase 2, we select one out of eighteen predefined layouts for slide generation. Hence, by design, each slide is constrained to have a maximum of 4 body elements apart from title, and other footer/meta elements. While this is not restrictive for business presentations that have limited element density per slide, lecture slides tend to show higher variability in number of body elements. Also, slide elements are significantly moved or resized often for better visibility or aesthetic appeal making them more unstructured than limited randomization performed in synthetic slides. This is evident from comparing element-wise spatial heatmaps of RealSlide benchmark and SynSlides dataset. Moreover, it has been noticed that lecture slides often use proprietary background templates Hence, synthetic slides might lack generalization to out-of-distribution lecture slides with high density of content or non-traditional layouts. An important future contribution to the pipeline is to implement an efficient method to assign randomized non-static layouts while minimizing overlap between elements to ensure high quality of synthetic slides.

C.2 Inadequate context while generation

Current method of context seeding in SynSlideGen is through a series of prompts injected at several stages of generation. While this method provides more opportunity to capture wide context compared to prior methods that use a single prompt, it still falls short of achieving content depth found in real lecture slides. Comparing RealSlide and SynSlides we find that real lecture presentations have, on average, three times the number of slides as compared to SynSlides. This difference directly affects content quality where real lecture presentations are found to be more intricate and advanced lecture content. We restrict synthetic slides to around 12 - 15 per presentation thereby effectively limiting the depth of content that can be covered per topic. Such biases may hinder use of SynSlides for other high-level document tasks like Visual Question Answering.

C.3 Separate annotation functions

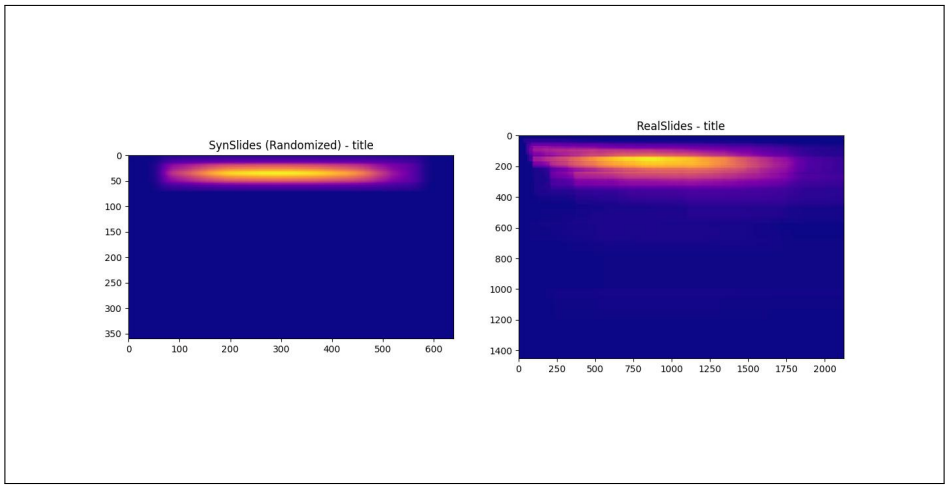
SynSlideGen provides automatic generation and annotation for two downstream tasks namely Slide Element Detection and Text-based Slide Image Retrieval. However, post generation of slide images, they annotation pipeline is required to be separate for each task. Hence, while the raw JSON content for each synthetic slide can be reused for multiple tasks, there requires additional effort to scale up number of downstream tasks for the same data.

C.4 Scaling synthetic slides

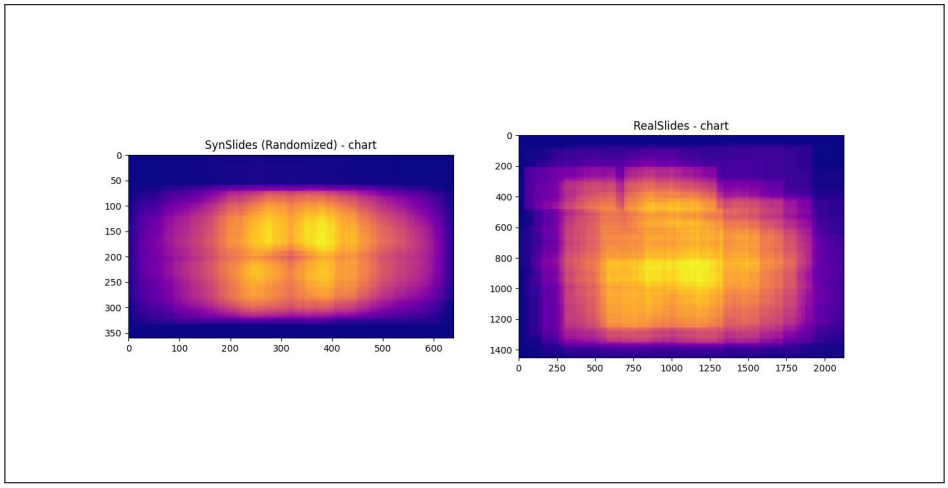
Our experiments with increasing synthetic slides for Slide Element Detection show that performance gains are marginal (see Table 2) and hence we opt to choose an optimal size set that balances performance and training time.

# Synthetic images for Training	Performance on RealTest (750 images)	
	Synthetic	Synthetic + Real
500	27.8	35.2
1000	28.1	36.4
2200*	29.8	38.8
4000	30.2	38.8
8000	30.7	39.7
16000	30.9	40.3

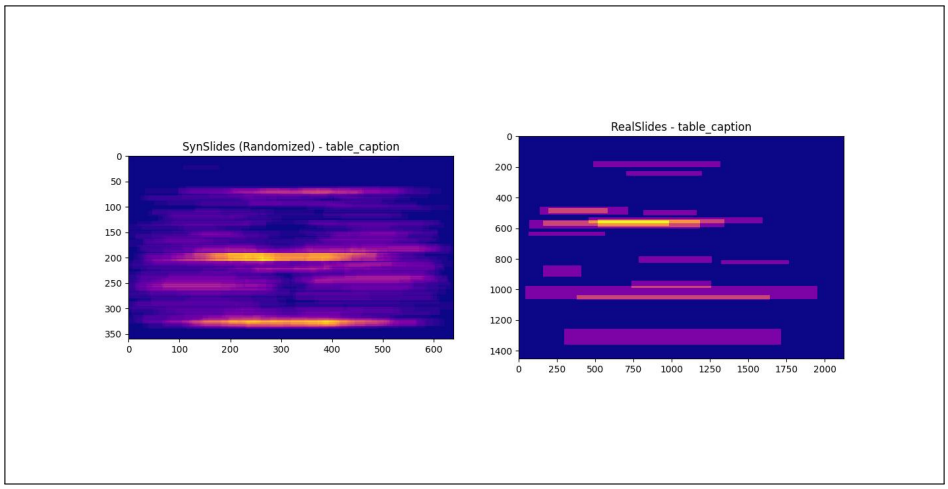
Table 2: Synthetic: model trained on synthetic images only. Synthetic + Real: model trained on synthetic, then 300 real images in two-stage finetuning setting. *Size refers to **SynDet**.



(a) Title

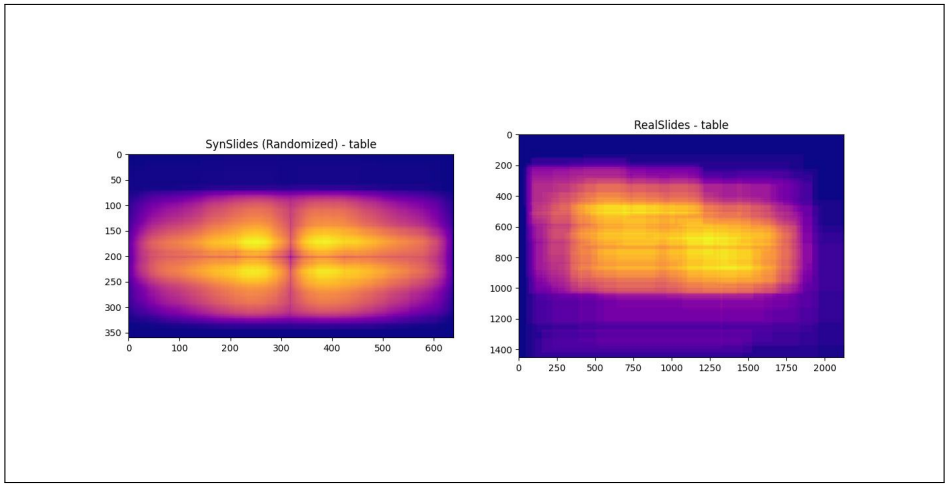


(b) Chart

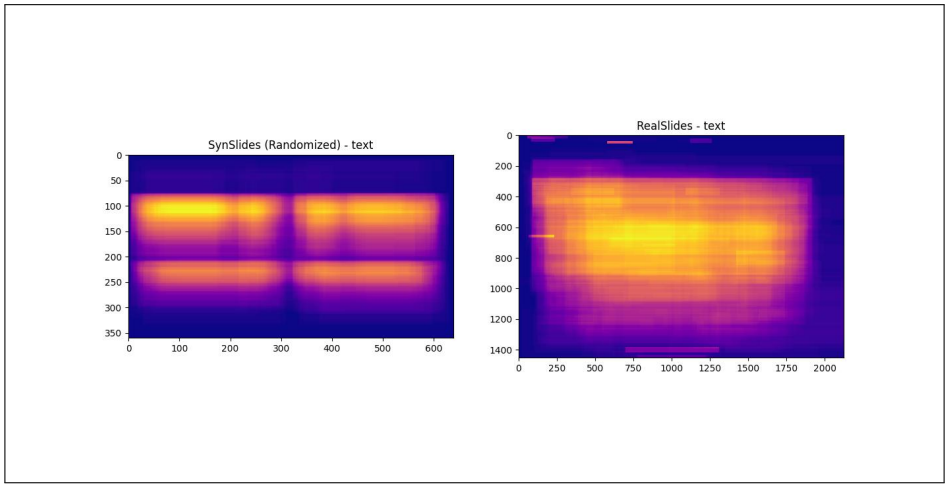


(c) Table Caption

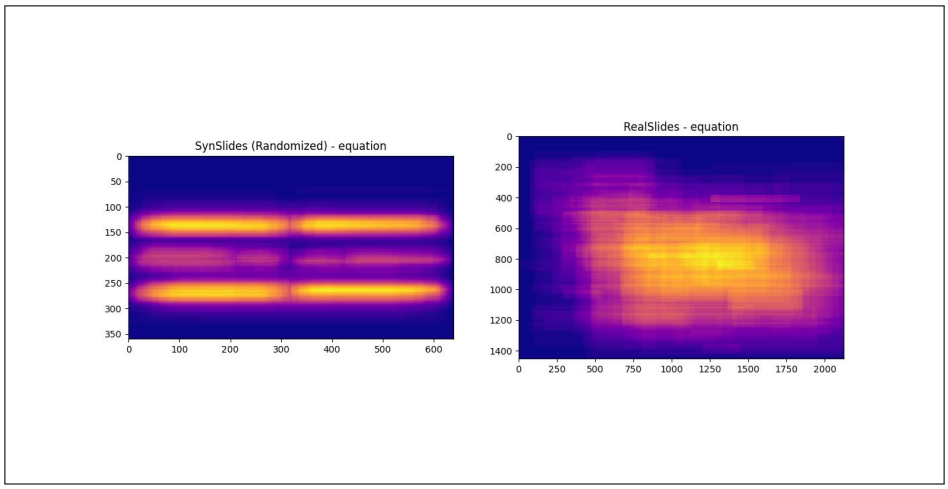
Fig. 4: Layout Heatmaps comparing randomized SynSlides (Left) with RealSlide layouts (Right).



(a) Table



(b) Text



(c) Equation

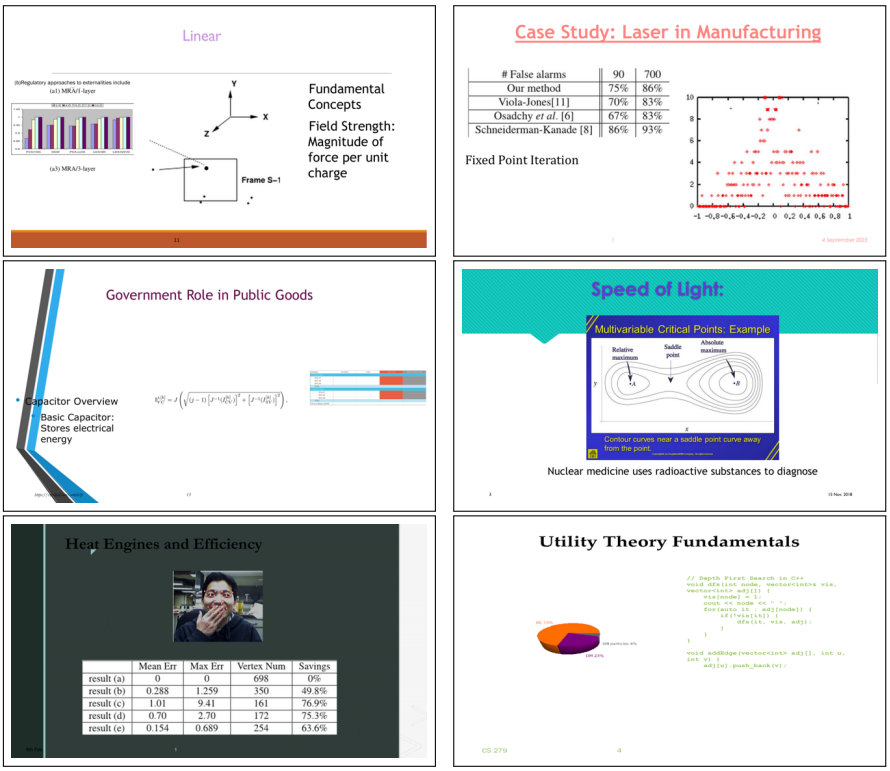
Fig. 5: Layout Heatmaps comparing randomized SynSlides (Left) with RealSlide layouts (Right).

D Discussion on Choice of Classes for Slide Element Detection

We define 16 classes for the task by merging and pruning classes from the 25-class SPaSe dataset. Specifically, we merge: Date, Footnote, Affiliation into Footer-Element; Presentation-Title and Slide-Title into Title; and Screenshot and Realistic Image into Image. We prune Map, Handwritten Equation, Comment, Drawing, and Legend due to insufficient instances in pre-made lecture slides. We determined the 16 classes based on their prevalence in lecture presentations, while also ensuring they are fine-grained enough for tasks such as slide narration systems.

E Visual Sample and Results

Here we present additional figures and tables that supplement the main text.



# False alarms	90	700
Our method	75%	86%
Viola-Jones[11]	70%	83%
Osadchy *et al.* [6]	67%	83%
Schneiderman-Kanade [8]	86%	93%

	Mean Err	Max Err	Vertex Num	Savings
result (a)	0	0	698	0%
result (b)	0.288	1.259	350	49.8%
result (c)	1.01	9.41	161	76.9%
result (d)	0.70	2.70	172	75.3%
result (e)	0.154	0.689	254	63.6%

Fig. 6: **SynDet** Visual Examples (Semantically Non Coherent Slides but Visually closer to Real Slides)

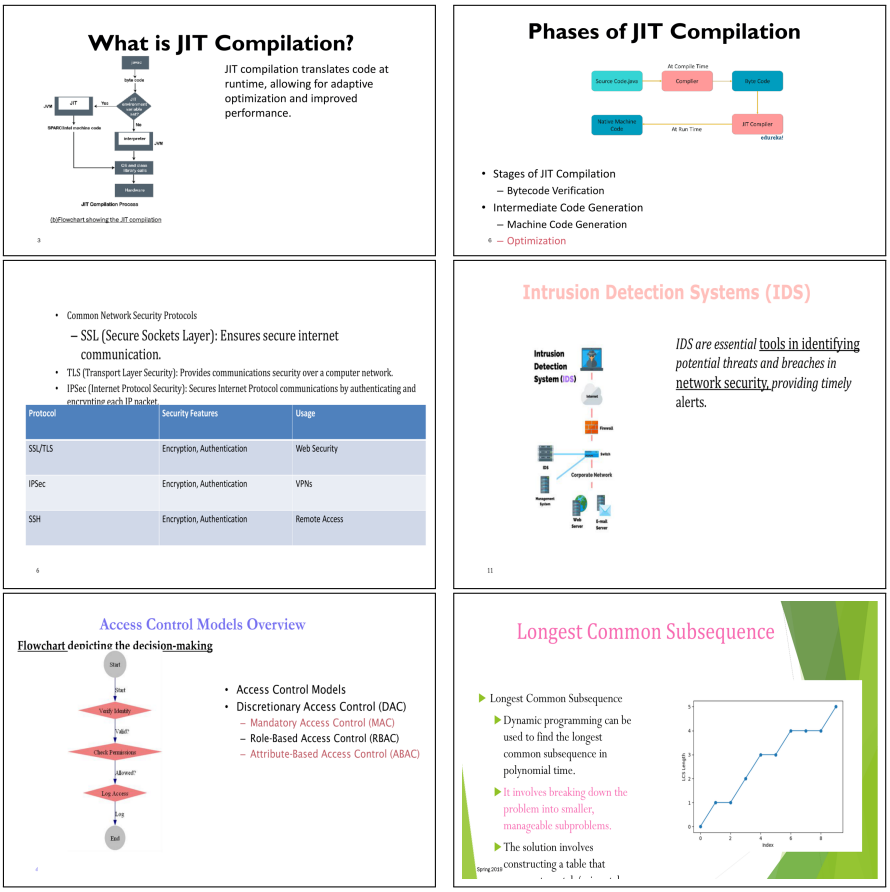


Fig. 7: **SynRet** Visual Examples (Coherent Slides)

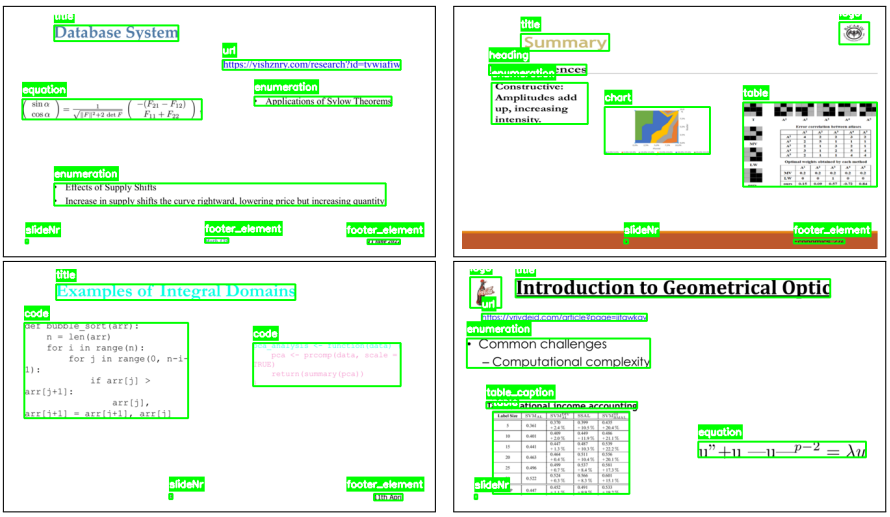


Fig. 8: Ground truth annotations of sample SynDet slide images

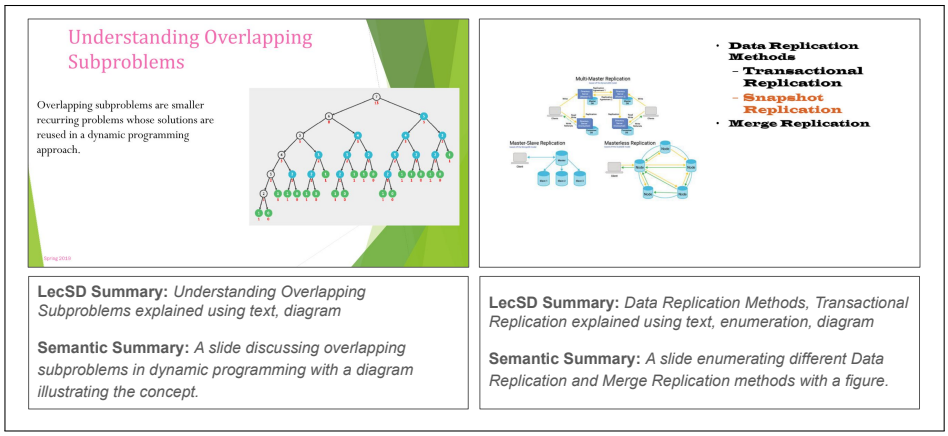


Fig. 9: Illustrate *semantic summary* and *LecSD-style summary* for two randomly selected sampled slides from our **SynRet**.

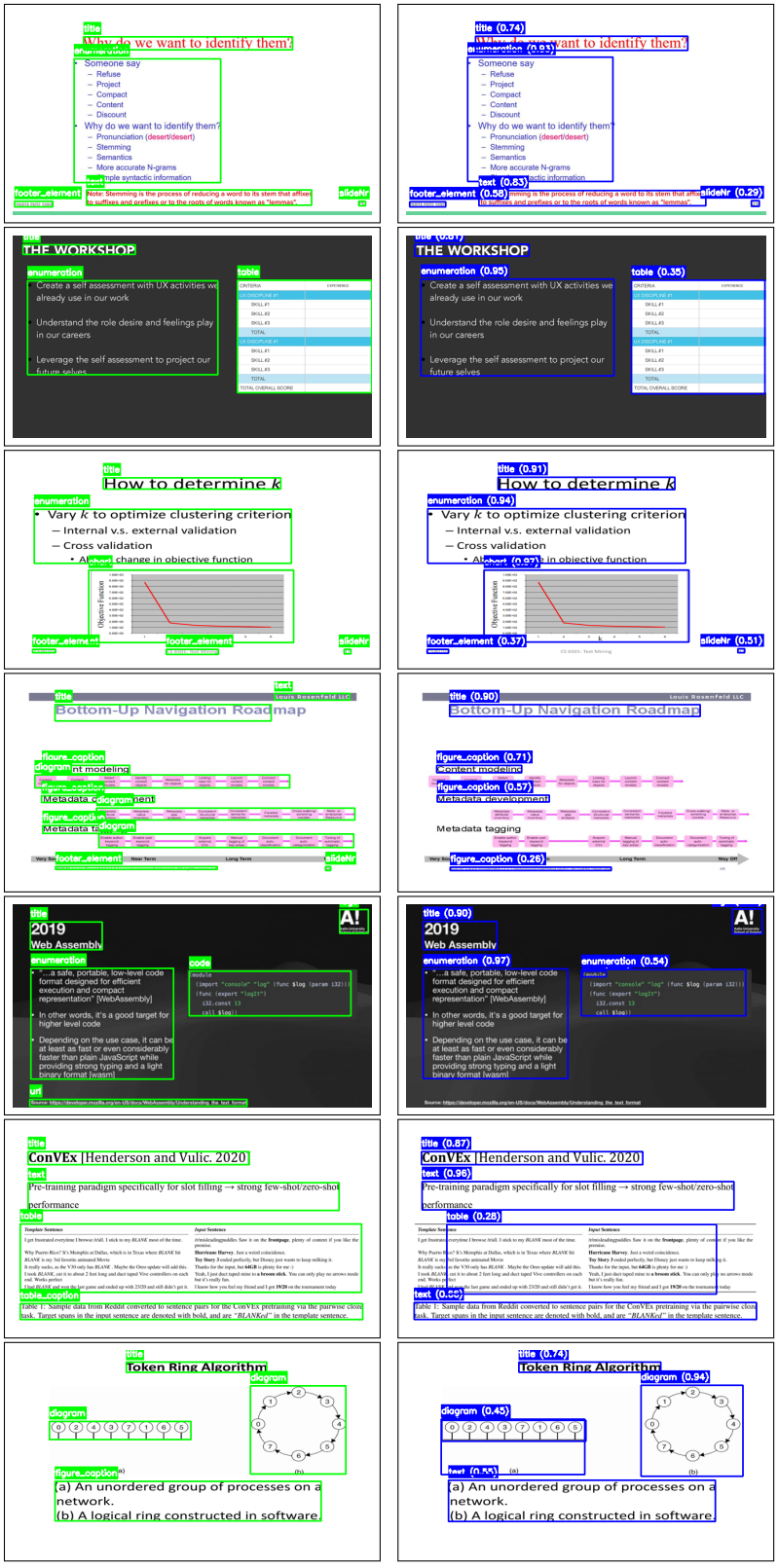


Fig. 10: Illustration of further selected visual results from YOLOv9 (Two-Stage), where green denotes ground truth bounding boxes and blue indicates predicted bounding boxes.(Later results highlight missed detections and class confusion)

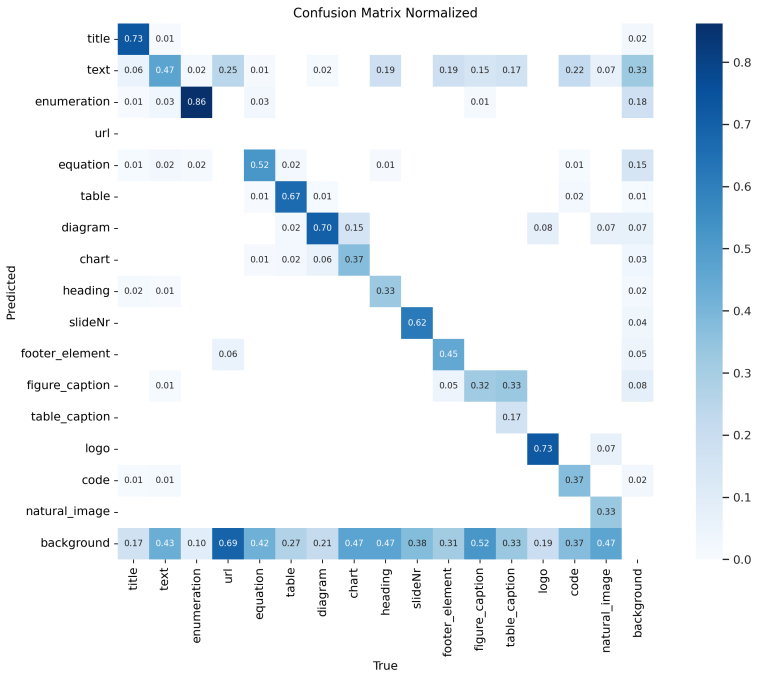
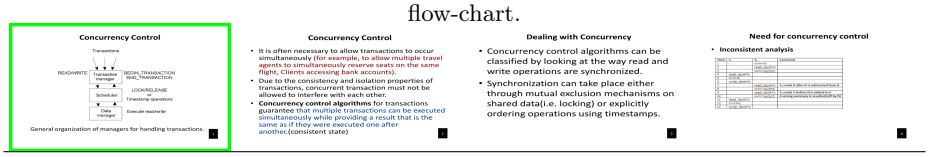


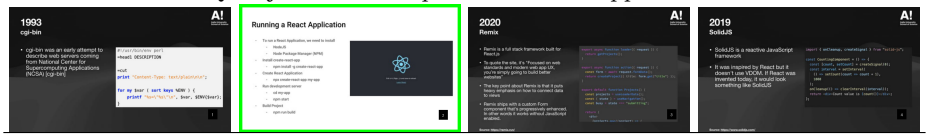
Fig. 11: Confusion Matrix for predictions of YOLOV9 Two stage finetuned model (SynDet+RealSlide Train) on RealSlide Test.

Query: Slide with a title and diagram about DynaEval.

Query: Concurrency Control Techniques explained using text, enumeration, flow-chart.



Query: Slide on steps to run React application.



Query: Lecture slide on k-means clustering and Expectation Maximization algorithm.

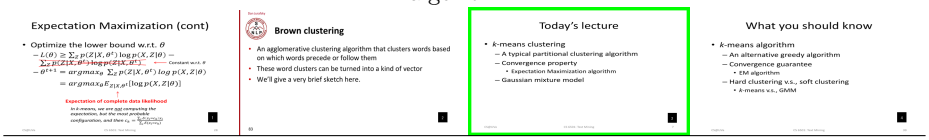


Fig. 12: Visual Analysis of result of CLIP model finetuned using SynRet data