Supplementary for Paper2Poster: Benchmarking Multimodal Poster Automation from Scientific Papers

A Ablation Study

We conduct ablation studies to evaluate three key design choices in PosterAgent: (1) the binary-tree layout strategy for layout planning; (2) the inclusion of a commenter module as a visual critic; and (3) the use of in-context examples to enhance the visual perception capabilities of the commenter.

We define the following variants:

- *Direct*: replacing the binary-tree layout with direct layout generation by an LLM;
- *Tree*: using the binary-tree layout strategy but removing the commenter module;
- *Tree + Commenter*: including the commenter module but without in-context examples;
- *Tree + Commenter + IC*: the full system, with both the commenter and in-context examples.

All ablation variants are implemented using `PosterAgent-4o`, keeping all other components unchanged to isolate the effect of each factor. We visualize and compare results across five randomly selected papers from Paper2Poster, as shown in Figures 1 to 5.

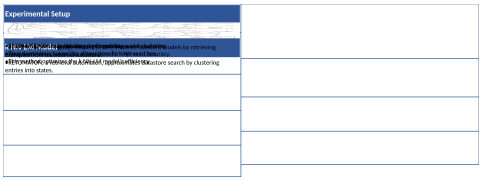
When prompting the LLM to directly generate poster layouts (*Direct*), the results are often structurally compromised (e.g., Figures 1a–3a), or resemble blog-style layouts that lack visual hierarchy and appeal (Figures 4a,5a). Fine-grained layout components, such as text boxes and figures, are especially challenging to synthesize in this setting: for instance, Figures1a–4a exhibit missing text boxes that leave noticeable blank areas, and Figure 4a fails to preserve the correct aspect ratio of figures.

The *Tree* variant, which omits the commenter module, leads to severe layout defects across all test cases (Figures 1b–5b), primarily manifesting as text overflow—where content spills outside its designated textbox or section panel—resulting in overlaps with other text or visual elements.

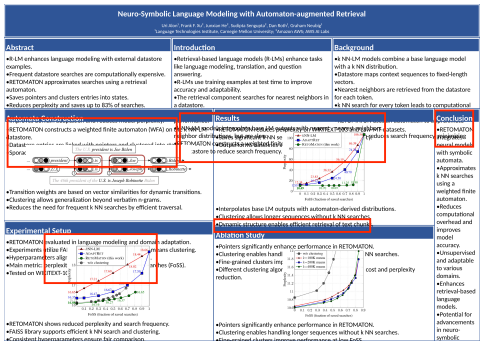
Using *Tree + Commenter*, which includes the commenter but without in-context examples, yields improved results compared to the variant without the commenter, but still exhibits noticeable issues. As shown in Figures 1c,2c,4c, and 5c, some degree of text overflow remains. Furthermore, Figures 3c and 4c highlight substantial unused white space that the commenter fails to flag in the absence of in-context guidance.

Finally, the full *Tree+Commenter+IC* system achieves the best results, as detailed throughout the main paper and demonstrated in Fig. 1d,2d,3d,4d.

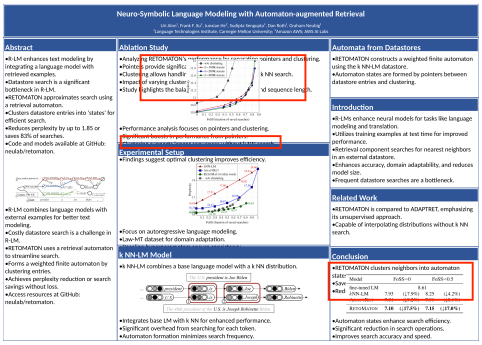
Submitted to 39th Conference on Neural Information Processing Systems (NeurIPS 2025). Do not distribute.



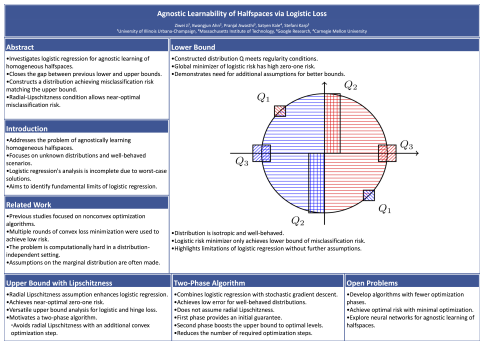
(a) Direct.



(b) Tree.

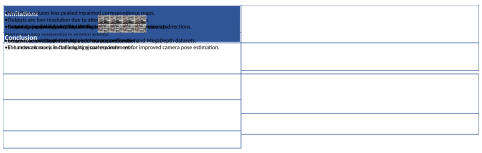


(c) Tree + Commenter.

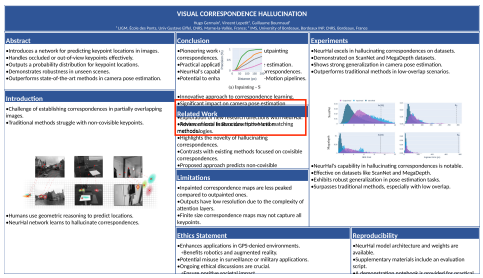


(d) Tree + Commenter + IC.

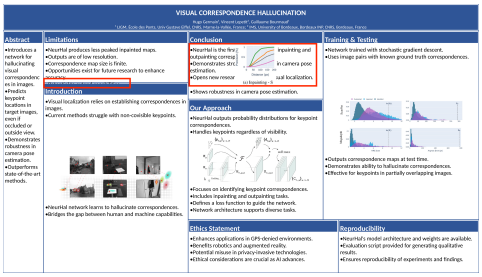
Figure 1: Ablation study on Agnostic Learnability of Halfspaces via Logistic Loss. Text overflow areas are highlighted with red bounding boxes.



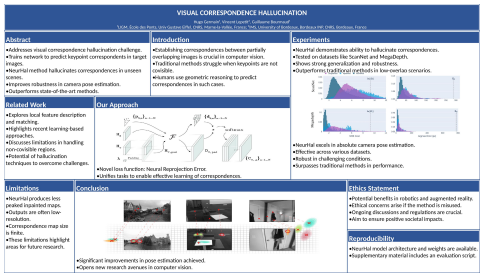
(a) Direct.



(b) Tree.

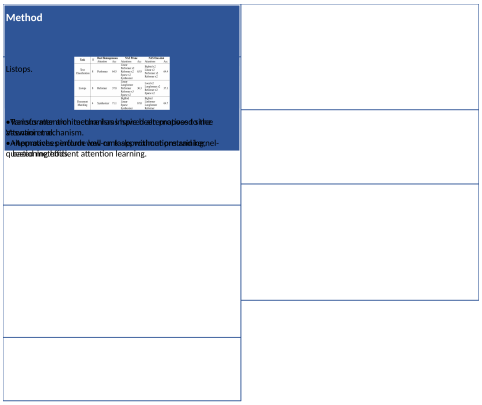


(c) Tree + Commenter.

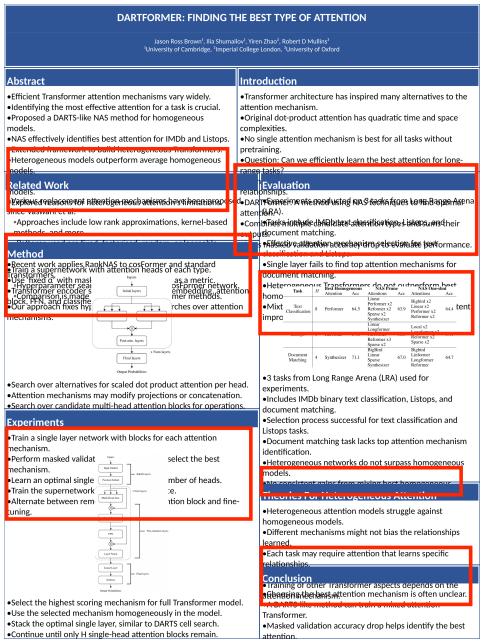


(d) Tree + Commenter + IC.

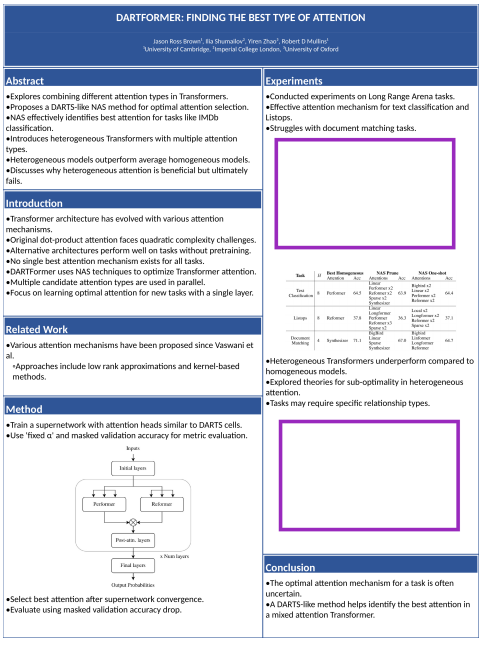
Figure 2: Ablation study on Visual Correspondence Hallucination. Text overflow areas are highlighted with red bounding boxes.



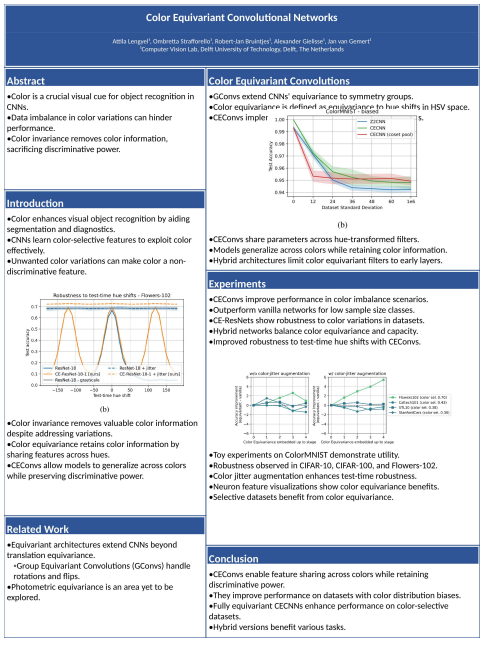
(a) Direct.



(b) Tree.



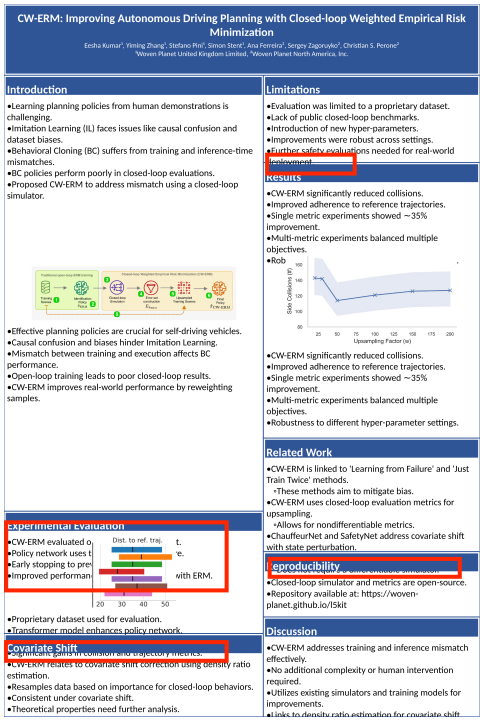
(c) Tree + Commenter.



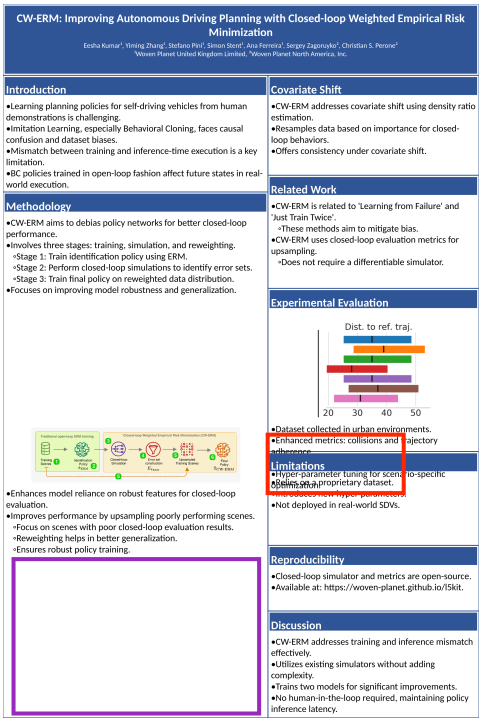
(d) Tree + Commenter + IC.

Figure 3: Ablation study on Color Equivariant Convolutional Networks. Text overflow areas are highlighted with red bounding boxes, large blank regions are highlighted with purple bounding boxes.

(a) Direct.



(b) Tree.



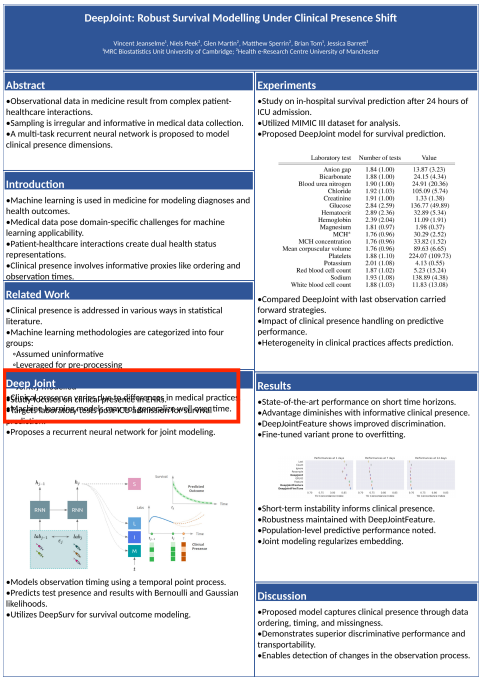
(c) Tree + Commenter.

(d) Tree + Commenter + IC.

Figure 4: Ablation study on CW-ERM: Improving Autonomous Driving Planning with Closed-loop Weighted Empirical Risk Minimization. Text overflow areas are highlighted with red bounding boxes, and large blank regions are highlighted with purple bounding boxes.

(a) Direct.

(b) Tree.



(c) Tree + Commenter.

(d) Tree + Commenter + IC.

Figure 5: Ablation study on DeepJoint: Robust Survival Modelling Under Clinical Presence Shift. Text overflow areas are highlighted with red bounding boxes.

B Additional Prompts

We present the prompts used by the planner module, covering three components: (1) the asset matching prompt; (2) the painter prompt; and (3) the commenter prompt.

Prompt: Asset Matching

System Prompt:
You are an expert assistant tasked with assigning images or tables to the most relevant poster sections. You will be given:

- JSON content of the poster outline, including each section's title and a brief description.
- A list of images (image_information) with captions and size constraints.
- A list of tables (table_information) with captions and size constraints.

Your goal is to produce a JSON mapping of each top-level section to exactly zero or one image/table that best fits that section's content. For each top-level section (named in the provided JSON "json_content"), decide:

- Whether an image or table (or none) is most relevant to the section's theme or description.
- If relevant, select the single most appropriate image or table to assign.
- Base this selection on the conceptual content described in the section ("research methods", "results", "conclusion", etc.) and compare it with the captions of the provided images or tables, choosing whichever fits best.
- If assigning an image, specify "image": <id>, where <id>is the identifier of the chosen image from "image_information".
- If assigning a table, specify "table": <id>, where <id>is the identifier of the chosen table from "table_information".
- Include an additional "reason" field briefly explaining why this assignment was made (e.g., how the image/table relates to the section content).
- If no image or table is assigned to a given section, omit that section from the final JSON (i.e., only list sections where you actually assign something).

Important Notes:

- The assignment should not be arbitrary. It must be logically consistent with the section's description and the provided caption for the image or table.
- Do not produce any layout properties or subsections here.
- The final output must be a single JSON object, mapping from section names to the chosen image/table ID plus the "reason" field.
- If multiple images or tables are suitable, select the single best one and assign only that.
- If "image_information" or "table_information" is empty, you may end up assigning nothing to any section.

Instructions:

1. Read and analyze the poster's top-level sections from {{ json_content }}.
2. Look at {{ image_information }} and {{ table_information }}. Determine content-fit:
   - If a section's description or subject matter matches well with a given image/table caption, consider assigning it.
   - If multiple images or tables seem relevant, choose the single best fit.
   - If none of the images or tables are relevant, or if none are provided, do not assign anything for that section.
3. Produce a single JSON object. Each key is the exact name of a top-level section (e.g., "Introduction", "Methods", "Results"), and the value is an object with:
   - `"image": image_id` or `"table": table_id`

- `"reason": short explanation describing why the image/table is assigned`

4. If no assignment is made for a section, exclude that section from the JSON.
5. No image can be reused for multiple sections. Each image/table can only be assigned to one section.
6. Ensure your final response strictly follows JSON syntax with no extra commentary.

Example Output Format:

```
{
  "Introduction": {
    "image": 1,
    "reason": "Image 1 depicts the central concept introduced
    in this section."
  },
  "Results": {
    "table": 2,
    "reason": "Table 2 summarizes the key metrics discussed
    in the results."
  }
}
```

Prompt: Painter

System Prompt:
You are an expert assistant tasked with producing bullet-point summaries for a given poster section. You will be given:

- A JSON object `summary_of_section` that contains:

```
{
  "title": "<section title>",
  "content": "<full text description>"
}
```

- An integer `number_of_textboxes`, which can only be 1 or 2.

Your goal is to produce a JSON object representing the bullet-point text for this poster section. Each "textbox" key (`textbox1` or `textbox2`) maps to a list of bullet-point entries. Each bullet-point entry must be a JSON object of the form:

```
{
  "alignment": "left",
  "bullet": true,
  "level": <indent_level>,
  "font_size": <integer>,
  "runs": [
    {
      "text": "<bullet point text>"
      # optionally "bold": true or "italic": true if needed
    }
  ]
}
```

Instructions:

1. If `number_of_textboxes` = 1, your final output must only have:

```
{
  "title": [ section title ],
  "textbox1": [ ... array of bullet items ... ]
}
```

2. If `number_of_textboxes = 2`, then you must produce *two* keys: `textbox1` and `textbox2`, and each must have the same number of bullet items. For example:

   ```
   {
     "title": [ section title ],
     "textbox1": [... N bullet items ...],
     "textbox2": [... N bullet items ...]
   }
   ```

   where both arrays have *identical length*.

3. Each bullet point is a JSON object with the structure shown above; you can create as many bullet points as needed (following the constraint about textbox count).
4. Make sure your final output is valid JSON, with no extra keys or additional formatting.
5. Return only the JSON object, nothing else.

Example Output:
Example when `number_of_textboxes = 1`:

```
{
  "title": [
    {
      "alignment": "left",
      "bullet": false,
      "level": 0,
      "font_size": 60,
      "runs": [
        {
          "text": "Methodology",
          "bold": true
        }
      ]
    }
  ],
  "textbox1": [
    {
      "alignment": "left",
      "bullet": true,
      "level": 0,
      "font_size": 48,
      "runs": [
        {
          "text": "Key point about domain-invariant component analysis."
        }
      ]
    },
    {
      "alignment": "left",
      "bullet": true,
      "level": 1,
      "font_size": 48,
      "runs": [
        {
          "text": "Supporting detail.",
          "bold": true
        }
      ]
    }
  ]
}
```

Example when `number_of_textboxes` = 2:

```
{
  "title": [
    {
      "alignment": "left",
      "bullet": false,
      "level": 0,
      "font_size": 60,
      "runs": [
        {
          "text": "Experimental results",
          "bold": true
        }
      ]
    }
  ],
  "textbox1": [
    {
      "alignment": "left",
      "bullet": true,
      "level": 0,
      "font_size": 48,
      "runs": [
        {
          "text": "Primary finding, bullet 1."
        }
      ]
    },
    {
      "alignment": "left",
      "bullet": true,
      "level": 0,
      "font_size": 48,
      "runs": [
        {
          "text": "Primary finding, bullet 2."
        }
      ]
    }
  ],
  "textbox2": [
    {
      "alignment": "left",
      "bullet": true,
      "level": 0,
      "font_size": 48,
      "runs": [
        {
          "text": "Additional commentary, bullet 1."
        }
      ]
    },
    {
      "alignment": "left",
      "bullet": true,
      "level": 0,
      "font_size": 48,
      "runs": [
```

```
        {
          "text": "Additional commentary, bullet 2."
        }
      ]
    }
  ]
}
```

Prompt: Commenter

System Prompt: You are an agent that is given three images:

- **Negative Example**: This image shows a bounding box with text overflowing outside it (i.e., text crossing or cut off by the box).
- **Positive Example**: This image shows a bounding box with text that fits completely (i.e., no text crossing or cut off).
- **Target Image**: This is the final image you must analyze.

From the first two images, you learn to interpret:

1. Whether text is overflowing (text crossing, cut off, or otherwise cannot fully fit in the box).
2. Whether there is too much blank space in the bounding box (i.e., the text is significantly smaller than the box, leaving large unused space).
3. Whether the text and bounding box are generally well-aligned (no overflow, no large blank space).

Then, for the **Target Image**, you must:

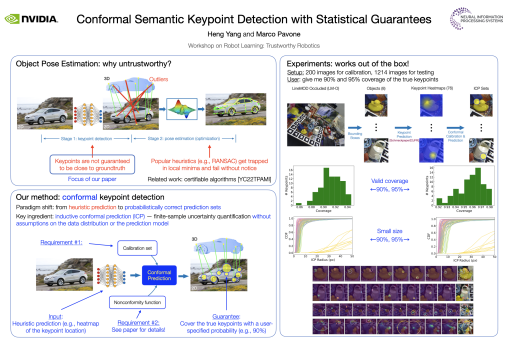
- If there is any overflow text, return `"1"`.
- If there is too much blank space, return `"2"`.
- If the text fits well (no overflow, no large blank space), return `"3"`.

Instructions:

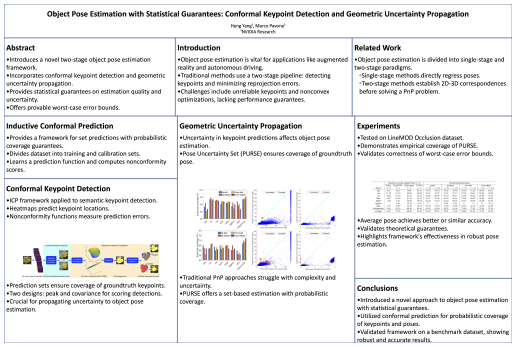
1. You are provided three images (negative example, positive example, and target).
2. Refer to the first two images (negative and positive examples) to understand:
   - What text overflow looks like
   - What too much blank space in a bounding box means
   - How a generally well-fitted bounding box appears
3. Analyze the third (Target) image's bounding box to check:
   - If there is overflow text, return `"1"`.
   - If there is too much blank space, return `"2"`.
   - Otherwise (if everything looks good), return `"3"`.

C Data Samples

We present additional examples of ground truth posters from Paper2Poster alongside the corresponding posters generated by PosterAgent.

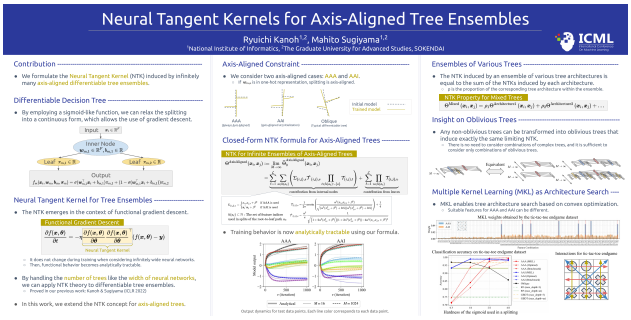


(a) Author-designed poster.

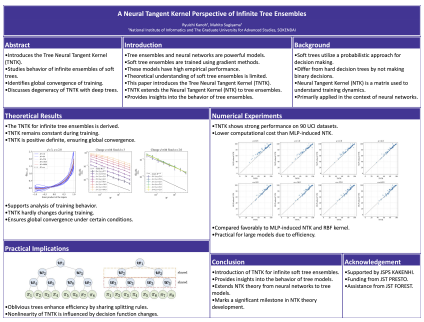


(b) PosterAgent-generated poster.

Figure 6: Posters for Conformal Semantic Keypoint Detection with Statistical Guarantees.

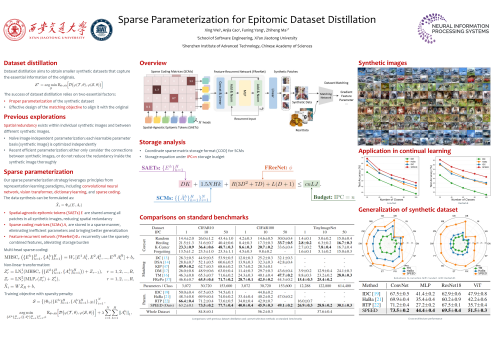


(a) Author-designed poster.

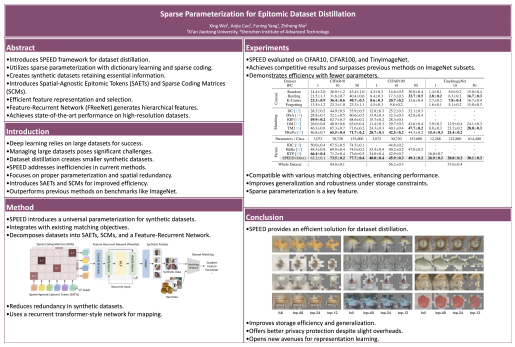


(b) PosterAgent-generated poster.

Figure 7: Posters for Neural Tangent Kernels for Axis-Aligned Tree Ensembles.

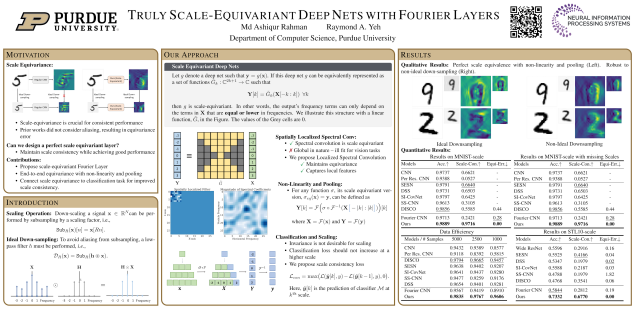


(a) Author-designed poster.

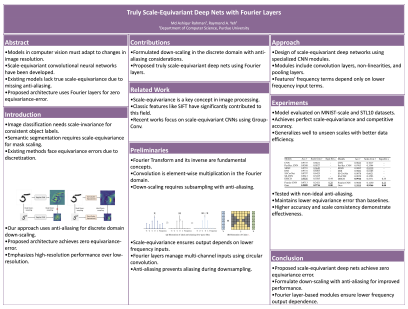


(b) PosterAgent-generated poster.

Figure 8: Posters for Sparse Parameterization for Epitomic Dataset Distillation.

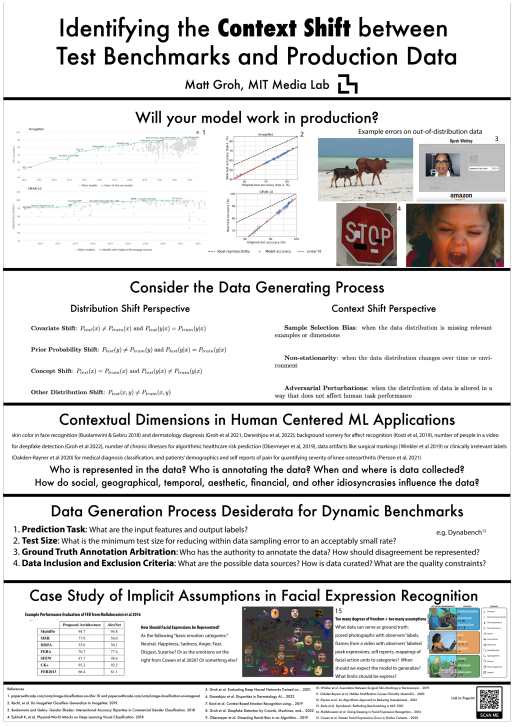


(a) Author-designed poster.

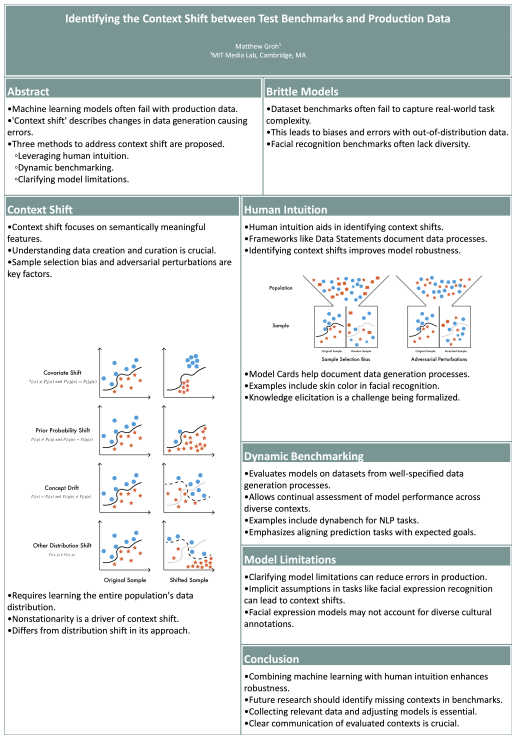


(b) PosterAgent-generated poster.

Figure 9: Posters for Truly Scale-Equivariant Deep Nets with Fourier Layers.



(a) Author-designed poster.



(b) PosterAgent-generated poster.

Figure 10: Posters for Identifying the Context Shift between Test Benchmarks and Production Data.

D Cost Analysis

Using `GPT-4o` as the backbone for both the LLM and VLM components, the average cost of generating a single paper with `PosterAgent-4o` is approximately:

$$\frac{98.1 \times 1000}{1,000,000} \times 5 + \frac{3 \times 1000}{1,000,000} \times 20 = 0.55 \text{ USD},$$

based on OpenAI's GPT-4o API pricing as of May 22, 2025.