1. Additional Architectural Details

1.1. Feature Extractors.

In this section, we provide additional details about the backbones in the proposed MONOMER. We discuss the hyperparameter choices in the Image Encoder, followed by the Text Encoder and finally the Layout Encoder. Please note that the hyperparameters are kept consistent across experiments on all the datasets.

1.1.1 Image Features

The image encoder is a DiT-backbone with encoder-only architecture having 4 layers, each containing 4 attention heads with model dimension of 512. The encoder takes 3 channel document image resized (using bi-cubic interpolation) to $224 \times 224$ resolution which is further cut into $16 \times 16$ sized patches and outputs a token sequence of length 197. The 197 tokens are formed as follows – $224 \times 224 + 1$, where the additional token corresponds to the CLS token as in the original BEiT [1]. We choose a pretrained DiT base model for our experiments that has a hidden dimension of 768. Since both query image $Q_{inp}$ and target image $T_{inp}$ are preprocessed to the same dimension, we obtain two feature vectors $Q_v, T_v$ each of size $BS \times 197 \times 1024$, where 1024 is the maximum sequence length and $BS$ denotes the batch size. Note that the maximum sequence length is a hyperparameter choice that is chosen based on the maximum number of text-blocks in the target document. The encodings are then padded to final vectors $Q_v, T_v$ of size $BS \times 1024 \times 1024$ each. The rationale behind doing so is to conveniently be able to perform the subsequent cross-attention with different modalities. We can summarize the sequence of operations as follows -

$$Q_v = \text{pad}(D(Q_{inp}^{inp})) \in \mathbb{R}^{1024 \times 1024}$$

$$T_v = \text{pad}(D(T_{inp}^{inp})) \in \mathbb{R}^{1024 \times 1024}$$

where $D$ is the DiT image encoder and $\text{pad}$ is the padding operation.

1.1.2 Text Features

We use a pretrained BeRT-based sentence transformer [5] that generates 768 dimensional embedding for a given block of text. The continuous blocks of text in the document are fed into this encoder to generate token sequence $T_{inp}^{text_t}$, $Q_{inp}^{text_q}$ of dimension $BS \times text_t \times 768$ and $BS \times text_q \times 768$ respectively, where $text_t$ is the number of text-blocks in the target document and $text_q$ is the number of text-blocks in the query patch. Additionally, we pad both $T_{inp}^{text_t}$, $Q_{inp}^{text_q}$ to a constant size of $BS \times 1024 \times 768$. Unlike all the other MONOMER parameters, the text encoder weights are kept frozen. Mathematically, text encoding is represented as follows –

$$Q_t = B(\text{pad}(Q_{inp}^{text_t})) \in \mathbb{R}^{1024 \times 768}$$

$$T_t = B(\text{pad}(T_{inp}^{text_q})) \in \mathbb{R}^{1024 \times 768}$$

where $Q_t, T_t$ are the final query and target text features and $B$ is the BeRT text encoder.

1.1.3 Bounding Box Features

We leverage a ViT-like [6] architecture to encode the bounding box (spatial) information in the target document and query patch. We implement an encoder-only transformer architecture with 4 layers, 4 heads and hidden dimension of 1024. It takes bounds of the target $T_{inp}^{text}$ and query $Q_{inp}^{text}$ of size $BS \times box_t \times 4$, $BS \times box_q \times 4$, where $box_t, box_q$ are the
number of bounding boxes in target and query respectively. Similar to the text-encoder, \( box_x \) and \( box_y \) are padded to the maximum sequence length of 1024. Weights of this encoder are initialized randomly. We denote the bounding box encoding as follows –

\[
Q_s = V(pad(Q_s^{inp})) \in \mathcal{R}^{1024 \times 1024} \tag{5}
\]

\[
T_s = V(pad(T_s^{inp})) \in \mathcal{R}^{1024 \times 1024} \tag{6}
\]

where \( V \) is the ViT-like bounding box encoder and \( Q_s, T_s \) are the final feature sets corresponding the query and target respectively.

1.2. Feature Fusion

1.2.1 Symmetric Attention Module.

The symmetric attention module consists of 2 multi-head attention (MHA) modules each containing 4 heads and embedding dimension of 512. To ensure that the input token feature dimension matches with MHA’s specifications, the input sequences are passed through fully-connected layers to project feature dimension onto dimension of 512. The outputs of the MHA blocks are concatenated (along last dimension) to obtain final token sequence with feature dimension of \( BS \times 1024 \times 1024 \).

1.2.2 Co-Attention and Cross-Attention Modules.

Co-Attention Module contains 3 symmetric attention modules one for each modality, outputting sequences \( VV, TT \) and \( SS \) of length 1024 and token size 1024.

\[
VV = SA(Q_v, T_v) \in \mathcal{R}^{1024 \times 1024} \tag{7}
\]

\[
TT = SA(Q_t, T_t) \in \mathcal{R}^{1024 \times 1024} \tag{8}
\]

\[
SS = SA(Q_s, T_s) \in \mathcal{R}^{1024 \times 1024} \tag{9}
\]

where \( SA \) is the Symmetric Attention operation.

Similarly, the Cross-Attention Module consists of 2 symmetric attention modules for generating spatio-visual features and 2 for attending text over those generated features. It generates \( S_qV_t \) and \( S_tV_q \) the dimensions of which are, once again, length of 1024 and token size of 1024. Finally we concatenate the outputs of Co-Attention and Cross-Attention blocks to create feature volume \( F_{sim} \). Obtain \( F_{sim} \) as follows –

\[
S_qV_t = SA(Q_s, T_v) \in \mathcal{R}^{1024 \times 1024} \tag{10}
\]

\[
S_qV_tT_t = SA(S_qV_t, T_t) \in \mathcal{R}^{1024 \times 1024} \tag{11}
\]

\[
S_tV_q = SA(T_s, Q_v) \in \mathcal{R}^{1024 \times 1024} \tag{12}
\]

\[
S_tV_qT_q = SA(S_tV_q, Q_t) \in \mathcal{R}^{1024 \times 1024} \tag{13}
\]

\[
F_{sim} = \text{concat}(VV, TT, SS, S_qV_tT_t, S_tV_qT_q) \tag{14}
\]

where \( F_{sim} \in \mathcal{R}^{1024 \times 1024} \) is the final set of features and are processed as described in the main paper.

1.3. Bounding Box Detection

The \( F_{sim} \) is passed through a linear layer followed by a sequence of 4 convolutional layers to produce features which are reshaped to give outputs at 4 different levels as described in the main paper. Then we apply a standard FPN [3] to obtain features at a common representation size of 1024. Finally, we generate proposals most similar to the query through an RPN [2] and subsequently detect bounding boxes using RoI Heads. We choose the default parameters for the RPN and RoI heads (from [4]) –

- RPN NMS threshold = 0.7
- RPN IOU threshold = 0.7 (FG), 0.30 (BG)
- RPN Score Threshold = 0
- ROI NMS Threshold = 0.40
- ROI Score Threshold = 0.05
- Detections per image = 200
- ROI IOU threshold = 0.50 (FG), 0.50 (BG)

where FG, BG are the foreground and background respectively. Note that the bounding box detection for LayoutLMv3 baseline is kept exactly the same as the proposed MONOMER whereas for BHRL, the github implementation is used.

1.4. Model size comparison vs baselines

On comparing the model sizes of the selected baselines with our proposed model, we found that BHRL has 48M parameters but is considerably worse at task performance compared to LayoutLMv3 (126M) and MONOMER (146M). Further, despite the comparable number of parameters in LayoutLMv3, our method outperforms it significantly (+20% in mAP).

2. Human Evaluation

In this section, we delineate the human evaluation conducted on the generated dataset through the proposed technique. A summary of the results have been tabulated in Table[1] We create 4 dataset split containing 40 samples each and share each split with 3 human evaluators to report the metrics. The high recall and precision over all the splits indicates that not only does our method generate high quality ground truths (87.96%) but is also able to find most of the target regions (81.07%) in a given document corresponding to a particular query. This saves a considerable amount of human annotation costs while maintaining reliability. Further, we also note that a substantial number of samples (48.12%) over all splits are complex and hard to search for in a document. While this metric is largely subjective,
<table>
<thead>
<tr>
<th>Metrics</th>
<th>Split-1</th>
<th>Split-2</th>
<th>Split-3</th>
<th>Split-4</th>
<th>Average over Splits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recall</td>
<td>87.25</td>
<td>71.93</td>
<td>80.74</td>
<td>84.38</td>
<td>81.07</td>
</tr>
<tr>
<td>Precision</td>
<td>83.06</td>
<td>87.83</td>
<td>91.41</td>
<td>89.58</td>
<td>87.96</td>
</tr>
<tr>
<td>F1</td>
<td>84.28</td>
<td>78.56</td>
<td>85.55</td>
<td>86.48</td>
<td>83.71</td>
</tr>
<tr>
<td>% Complex Pattern</td>
<td>43.33</td>
<td>39.16</td>
<td>80.00</td>
<td>30.00</td>
<td>48.12</td>
</tr>
<tr>
<td>% non exact but similar matches over correct matches highlighted.</td>
<td>79.61</td>
<td>93.07</td>
<td>91.09</td>
<td>86.17</td>
<td>87.48</td>
</tr>
</tbody>
</table>

Table 1: Human Evaluation of the proposed dataset over 4 different splits. The last row indicates the non-trivial nature of dataset generation through the percentage of examples that are not exactly the same as the query.

<table>
<thead>
<tr>
<th>Heuristic</th>
<th>AP50</th>
<th>AR50</th>
<th>AP75</th>
<th>AR75</th>
<th>mAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heuristic</td>
<td>89.94</td>
<td>41.70</td>
<td>64.81</td>
<td>29.89</td>
<td>44.39</td>
</tr>
<tr>
<td>MONOMER</td>
<td>81.59</td>
<td>59.88</td>
<td>73.17</td>
<td>52.78</td>
<td>56.49</td>
</tr>
</tbody>
</table>

Table 2: Performance comparison between data generation heuristic as a baseline and MONOMER on human-annotated data

Consistency over multiple splits verifies our claim. Finally, the last row shows that query matching within targets is non-trivial such that only 12.42% cases where snippet and highlighted similar regions are exact matches with the rest of the non-trivial matches containing the same layout but possibly different variation, text, fonts etc which allows the model to learn “advanced one-shot search capabilities”.

**Hard samples (question 3 in human evaluation)** The evaluators were asked to decide if a snippet is hard based on whether its structure is complex - a) the snippet comprises of a significant number of elements or that the elements are arranged in a complex layout. Further, they were instructed to judge a snippet as hard b) if they would find it difficult to search a given query snippet in some target document.

3. Dataset Generation

**Link with real world data** We asked the human evaluators to annotate 160 test samples (used in human study) to compare the performance of our data generation heuristic (as baseline) to that of our proposed MONOMER. The results in Table 2 indicate that MONOMER performs better than heuristic and demonstrates its ability to perform well on a real dataset, highlighting the effectiveness of the proposed data generation technique to enable MONOMER to generalize better. It is apparent in comparison with the heuristic-based baseline in Table 2 that training our model on this dataset enables us to obtain a more generalized one-shot detector. We hypothesize that despite our heuristic not finding an exhaustive set of matches (81% recall) because it is more stringent and based on layout-based matching, the data generated by our dataset generation heuristic contains enough accurate samples for the model to learn snippet similarity. We also observe in Table 2 that the average precision (@50) of the heuristic remains high, and recall is slightly lower, verifying our hypothesis. Further, at a higher threshold of 0.75, MONOMER performs better in AP75, AR75, and mAP.

**Limitations of dataset generation** It is noteworthy to mention that the dataset generation method relies on a heuristic (metric Eq. 1 in main paper) and consequently encounters some failure cases. In this section, we discuss some of these failure cases using qualitative examples. The quantitative efficacy of our method is already elucidated in Table 1. A summary of failure cases is presented in Fig. A-Fig. B. Particularly, in Fig A the crop only contains a single “Yes/No” choice option, however it considers elements in the target with multiple structures within it. This happens as a result of the threshold penalty and a less strict size heuristic. The additional elements are considered as single blocks of text or fillable areas which our metric fails to capture. Further, in Fig B the heuristic is unable to discern the difference between a block of comma separated text and option-like text. This can be attributed to lack of detailed annotations (of low level elements) in the dataset, typically “option-like” text is annotated as a list which wasn’t correctly done in this instance. Lastly, in Fig C the heuristic fails to consider the length of the text and ends up matching a large block with a smaller target. This can be regulated using the size penalty. However, we reiterate that these examples occur sporadically and the same is justified through a quantitative analysis in Table 1.

4. Additional Qualitative Results

We add more results produced by the proposed MONOMER in this section. Please refer to Fig. 3 to Fig. 7 for the predictions.

References

Figure 1: False positive examples (1)
2021.


Figure 2: False positive examples (2)
Figure 3: Flamingo Forms Examples (1)
Figure 4: Flamingo Forms Examples (2)
Figure 5: Flamingo Forms Examples (3)
Figure 6: PubLayNet Examples (1)
Figure 7: PubLayNet Examples (2)