A. Pseudocode Algorithm

The pseudocode for integrating CLIPTER into a recognizer is presented in Algorithm 1. This algorithm outlines the key components of our method, including image encoding, pooling, fusion mechanism, and integration point that divides the recognizer into encoder and decoder. In particular, the algorithm highlights that the image encoding operation is performed only once per image, regardless of its word count, and can be executed in parallel with the detection operation.

Algorithm 1: CLIPTER PyTorch-like pseudocode

```python
# image encoding (in parallel to detection)
with torch.no_grad():
    img_f = img_encoder(img)  # (1 + HW, d)
    img_f = [img_f[0], avg_pool2d(img_f[1:], k)]
preds = []
for crop in text_crops:
    # recognizer encoding
    crop_f = recog_encoder(crop)
    # fusion by gated cross attention
    merged_f = fusion_ca(query=crop_f, key=img_f, value=img_f)
    c = torch.tanh(alpha)
    fused_f = (1 - c) * crop_f + c * merged_f
    # recognizer decoding
    preds.append(recog_decoder(fused_f))
```

B. Datasets

Our work utilizes a highly-diverse collection of 13 public benchmarks, depicted in Fig. 1 and Fig. 2. Since CLIPTER relies on the whole image together with the cropped words, we use datasets that have recognition and detection annotations, usually intended for the task of end-to-end text spotting. Therefore, we could not utilize some public test sets which contain only full images without localization annotations or cropped words without the full images. To mitigate this, we evaluate our method in these cases on the validation set or part of the training set. Nevertheless, we needed to omit IIIT-5k [16] which contains only cropped text images and CUTE-80 [18] which does not contain end-to-end annotations. Below, we describe our data pre-processing and then, provide details on each dataset.

B.1. Data Pre-Processing

Our work applies the same data filters on all datasets. In particular, we filter out words with the flag of illegible and words that have ignore labels, i.e., "#", "##", "###", "####" in general, ":" in TextOCR, and "*" in Uber. From the training data, we follow [5] and also exclude text that consists of non-alphanumeric characters, long words that contain more than 25 characters, and vertical text by filtering words with more than two characters that their image height is greater than their image width.

B.2. Dataset Details

Below, we provide general details on each dataset and describe our data split into train, validation, and evaluation sets. A summary of these splits appears in Tab. 1, containing also data sizes. As we work on entire images as well as crops, we perform the splits at the entire image level.

ArT[8] is a dataset of arbitrary-shaped text, collected from the train set of Task 3[1]. The train set is divided into 80% for training, 10% for validation, and 10% for evaluation.

COCO-Text[23] is based on COCO dataset[2], containing text in natural images[3]. We consider the training and validation sets that are published with bounding boxes, and split

---

1. https://rrc.cvc.uab.es
2. https://cocodataset.org
Figure 1: **Datasets Part 1.** We provide examples from each of the datasets used in this work.
Figure 2: **Datasets Part 2.** We provide examples from each of the datasets used in this work.
Figure 3: **Low Data Regime – TRBA & ViT-STR.** We evaluate the effect of CLIPTER with limited training data on TRBA [4] (left) and ViTSTR [3] (right). Roughly speaking, adding CLIPTER to these architectures has more impact than doubling the training data amount in terms of reducing the error rate.

---

the training set into 90% for training and 10% for evaluation.

HierText[15] features hierarchical annotations of text in natural scenes and documents⁴. We consider the training and validation sets which have available bounding boxes, and split the training set into 90% for training and 10% for evaluation. In this dataset, we filtered words that are annotated as vertical.

IC13[14] contains images that are focused around the text content¹. Since only the training set is provided with full annotations, we use it all for evaluation.

IC15[13] contains incidental scene text and therefore is more challenging¹. The test set here is the official one, while the training set is divided into 90% for training and 10% for validation.

LSVT[22] contains scene text in street view images¹. Here, only the training set has full annotations. Therefore, we divide it into 80% for training 10% for validation, and 10% for evaluation.

MLT19[17] is a multilingual dataset¹. The training set is divided into language subsets, from which we consider English, French, German, and Italian. We split these data into 80% for training, 10% for validation, and 10% for evaluation.

OOV[10] is a new dataset containing out-of-vocabulary scene text¹. Since this dataset is based on other datasets, we did not use its training set, but use its validation set for evaluation. In this dataset, we filter words that are annotated as non-English or vertical.

---

⁴https://github.com/google-research-datasets/hier-text

---

RCTW[20] is a dataset for reading Chinese text in images⁵. We split the published training set in 80% for training, 10% for validation and 10% for evaluation.

ReCTS[25] contains Chinese text on signboard¹. We split the published training set in 80% for training, 10% for validation and 10% for evaluation. In this dataset, we ignore words that are annotated with the flag of ignore.

SVT[24] contains street view text in images from Google Street View⁶. Here, we use the official test set and divide the

---

⁴https://github.com/google-research-datasets/hier-text

⁵https://rctw.vlrlab.net

⁶https://tc11.cvc.uab.es/datasets/SVT_1
Table 2: Cross-Attention Model Size.

<table>
<thead>
<tr>
<th>CA Model</th>
<th># Attention Heads</th>
<th># Hidden Layers</th>
<th>Hidden Size</th>
<th>Intermediate Size</th>
<th># Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gated-Attention</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>328K</td>
</tr>
<tr>
<td>MH-CA Tiny</td>
<td>2</td>
<td>2</td>
<td>128</td>
<td>512</td>
<td>923K</td>
</tr>
<tr>
<td>MH-CA Mini</td>
<td>4</td>
<td>4</td>
<td>256</td>
<td>1,024</td>
<td>5.3M</td>
</tr>
<tr>
<td>MH-CA Small</td>
<td>8</td>
<td>4</td>
<td>512</td>
<td>2,048</td>
<td>18.1M</td>
</tr>
</tbody>
</table>

training set into 90% for training and 10% for validation. **TextOCR**[21] contains high quality images from OpenImages with an average of 30 words per image. Here, we use the published validation set and divide the training set into 90% for training and 10% for evaluation. **Uber**[26] contains street-level images collected from car mounted sensors. We keep the original split of training, validation, and evaluation sets.

C. Implementation Details

Multi-head Cross-Attention fusion mechanism. Our implementation of the Multi-Head Cross-Attention (MH-CA) mechanism is based on the implementation of BERT[9, 7] proposed by HuggingFace. Table 2 presents further architectural details.

Training details. Baseline STR models are trained with the hyperparameters published by respective authors. CLIPTER is trained for 20 epochs with a learning rate varying from $1 \times 10^{-5}$ to $3 \times 10^{-5}$. Specifically, gated-attention, MH-CA tiny, mini and small are trained with learning rates of $2 \times 10^{-5}$, $3 \times 10^{-5}$, $3 \times 10^{-5}$ and $1 \times 10^{-5}$ respectively.

D. Low Data Regime

Similarly to analysis performed in the main paper over PARSeq, we evaluate the effect of our method in the low data regimes on TRBA and ViTSTR architectures. As shown in Fig. 3, utilizing CLIPTER on these schemes achieves better results than the baseline model with doubled amount of training data.

E. Latency Analysis

To evaluate the impact of our solution on recognition latency, we conduct end-to-end (E2E) experiments on the ICDAR-15 and Total-Text datasets, and calculate the frames per second (FPS). To this end, we use the ResNet50-based detection model from GLASS [19][10] and exclude their recognition components. Our experiments are conducted on a single V100 NVidia GPU and a simple PyTorch implementation, without any optimizations, such as TensorRT, that could improve the latency results. We calculate the latency using PyTorch benchmarking code[11], with FPS calculated as the average of the median run-time per image. Evaluation metrics are in accordance with the protocol of [19].

F. Additional Experiments

F.1. Synthetic Data

In this part, we aim to analyze the effect of utilizing synthetic data. To this end, we train PARSeq with and without CLIPTER also on the large synthetic datasets of MJ [12] and ST [11]. As shown in Tab. 3, adding the large synthetic data, about 14M images, to the training set only marginally improves the results, indicating on the low impact of synthetic data when there is a lot of real-world data. That said, these datasets do lead to significant improvements on IC13 and IC15. This finding, revealed also in [1, 2], indicates that these datasets mainly represent specific types of natural scenarios.

F.2. Breaking-Down Results on Uber-Text

We utilize Uber-Text[26] word categories to break down the results of PARSeq with and without CLIPTER. As shown in Tab. 4, our method is especially efficient on business name (+1.3%) and street numbers (+1.3%). We believe that these improvements are thanks to the use of a vision-language model that was pretrained also on the textual descriptions of the images, which often contain such information as it is crucial for understanding the scene.

F.3. Dense Documents

We conduct both a quantitative (Figure 5) and qualitative (Figure 4) analysis on the text-dense HierText dataset. The results demonstrate that our model consistently improves accuracy, even in highly text-dense images with over 100 words.

G. Further qualitative analysis

Fig. 6 displays additional examples showcasing benefits of CLIPTER.

---

7https://storage.googleapis.com/openimages/web/index.html
8https://textvqa.org/textocr
10https://github.com/amazon-science/glass-text-spotting
11https://pytorch.org/tutorials/recipes/recipes/benchmark.html#pytorch-benchmark
Figure 4: **Quantitative Results on Rich-in-Text Images.** Images with dense text (>100) that benefit from integrating scene-level information using CLIPTER. Green boxes highlight words accurately transcribed by PARSeq+CLIPTER but not by PARSeq, while red boxes indicate the opposite.

<table>
<thead>
<tr>
<th>Method</th>
<th>SVT</th>
<th>IC13</th>
<th>IC15</th>
<th>COCO</th>
<th>RCTW</th>
<th>Uber</th>
<th>ArT</th>
<th>LSVT</th>
<th>RECTS</th>
<th>MLT19</th>
<th>TextOCR</th>
<th>HierText</th>
<th>Average</th>
<th>Weighted Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>PARSeq [6]</td>
<td>96.1</td>
<td>98.9</td>
<td>85.7</td>
<td>80.5</td>
<td>81.4</td>
<td>83.2</td>
<td>91.2</td>
<td>80.2</td>
<td>91.8</td>
<td>91.5</td>
<td>85.2</td>
<td>87.4</td>
<td>87.8</td>
<td>85.6</td>
</tr>
<tr>
<td>+ CLIPTER Vision</td>
<td>96.6</td>
<td>99.1</td>
<td>85.9</td>
<td>81.0</td>
<td>82.1</td>
<td><strong>84.4</strong></td>
<td><strong>91.7</strong></td>
<td><strong>81.8</strong></td>
<td>91.8</td>
<td>91.6</td>
<td><strong>86.0</strong></td>
<td>88.0</td>
<td>88.3</td>
<td><strong>86.4</strong></td>
</tr>
<tr>
<td>∆</td>
<td>+0.5</td>
<td>+0.2</td>
<td>+0.2</td>
<td>+0.5</td>
<td>+0.7</td>
<td>+1.2</td>
<td>+0.5</td>
<td>+1.6</td>
<td>0</td>
<td>+0.1</td>
<td>+0.8</td>
<td>+0.6</td>
<td>+0.5</td>
<td>+0.8</td>
</tr>
</tbody>
</table>

Table 3: **Accuracy on Scene Text Benchmarks With and Without using Synthetic Data.** Utilizing the large synthetic datasets of MJ [12] and ST [11] improves performance on the more common benchmarks of SVT, IC13, and IC15. However, the averaged performance across all datasets is marginally better due to the existence of many real-world images.

<table>
<thead>
<tr>
<th>Street Number</th>
<th>Business Name</th>
<th>Street Name</th>
<th>None</th>
<th>Street Number Range</th>
<th>Secondary Unit Designator</th>
<th>Phone Number</th>
<th>Traffic Sign</th>
<th>License Plate</th>
</tr>
</thead>
<tbody>
<tr>
<td>22,701</td>
<td>14,254</td>
<td>5,885</td>
<td>4,866</td>
<td>1,708</td>
<td>98</td>
<td>32</td>
<td>16</td>
<td>1</td>
</tr>
<tr>
<td>Parseq</td>
<td>78.3</td>
<td>85.7</td>
<td>95</td>
<td>82.4</td>
<td>96.3</td>
<td>86.7</td>
<td>50</td>
<td>93.8</td>
</tr>
<tr>
<td>+ CLIPTER Vision</td>
<td>79.6</td>
<td>87</td>
<td>95.4</td>
<td>83.7</td>
<td>96.5</td>
<td>88.8</td>
<td>46.9</td>
<td>93.8</td>
</tr>
<tr>
<td>∆</td>
<td>+1.3</td>
<td>+1.3</td>
<td>+0.4</td>
<td>+1.3</td>
<td>+0.2</td>
<td>+2.1</td>
<td>-3.1</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 4: **Accuracy on Uber-Text per Word Category.** The number of words in each category is listed below its name. CLIPTER is mostly effective on street numbers and business names, often critical information for scene understanding.
Figure 5: Enhancing Performance in Dense-Text Images. This figure illustrates the averaged improvement in accuracy and the number of accurately transcribed words relative to the total number of words in the image. Our algorithm demonstrates remarkable success even in densely-packed text images.
Figure 6: **Positive flips.** Examples in which CLIPTER corrected the prediction of PARSeq and matched the GT annotation.
Figure 7: **Negative flips.** Examples in which CLIPTER harmed the prediction of PARSeq which previously matched the GT annotation.
References


