A. Baseline Post-Processing Operations

In order to be able to compare the generic baselines with STEP, we had to couple them with post-processing operations. The biggest challenge was regarding queries with spaces, since generic models are usually trained at word-granularity level. The post-processing operations performed during validation and test differ from each other due to the nature of the queries and text found in them.

A.1. Validation Split

Our validation set contains queries with either 1 space or no spaces. When the query contains no space, all instances that do not match the queried regular expression get removed. The rest of the instances are considered positive instances.

When a query contains a space we split it into two sub-queries. Each sub-query is one side of the query, the space being the splitting point. The output instances of the model are filtered out if they do not match one of the two sub-queries. Once the instances are filtered, we merge pair of instances that match the sub-queries and are closer than a certain threshold. We merge both the polygons of the instances and the translation with a space between transcriptions. Figure 1 shows how these post-processing operations are performed on an example from the validation set.

A.2. Test Split

The queries featured in the test split can contain more than one space, as seen in the example formats in Table 2. Furthermore, some instances feature arbitrary separations between the different characters of the code that do not follow any specific format (see examples in Figure 6), which can cause unpredictable detection fragmentations. See for example the TESTR's row in Figure 3, where the TARE and UIC codes have been fragmented at arbitrary points.

In order to merge the detections under such circumstances, we have opted for a different strategy. First, we check if any of the unmerged instances match the query. Next, we iteratively merge pairs of instances that are close to each other (within a certain threshold). After each iteration, we check if the newly merged pairs match the query. The process ends when we can not merge any more instances. Instances that have been matched are not merged with more instances in the next iteration. This process is illustrated in Figure 2.

B. Baseline Training Details

In all the baselines we use pre-trained weights provided by the authors to initialize the networks and fine-tuned on the vanilla HierText training split. Unless explicitly stated, we use the default training settings used in the original implementations.

On ABCnet v2 [3] we use the provided pre-trained weights on their SynthText 150k dataset and MLT 17. We converted the HierText polygonal annotations to the ABCnet bezier annotations. We fine-tune on HierText for 50k iterations with an initial learning rate of 10^{-3}, which is decayed by a factor of 0.1 at 20k and 40k iterations and a batch size of 9 images. Since HierText features some images with dense text, we increase the maximum number of proposals per image to 300.

For SwinTextSpotter [2] we use the provided pre-trained weights trained on MLT 17 and the SynthText 150k datasets. We fine-tune the model on HierText for 35k iterations with an initial learning rate of 10^{-4}, decayed by 0.1 at step 35k, and a batch size of 4. We also increase the number of proposals per image to 300.

Finally, we use the pre-trained weights provided by the authors of TESTR which contains, as we said, the SynthText 150k, MLT 17 [4] and Total-Text datasets. The model is trained for 30k iterations with an initial learning rate of 10^{-4}, a learning rate decay of 0.1 at iteration 25k, and a batch size of 6 images. The maximum number of queries is increased to 300.

C. Architecture Details

STEP uses a ResNet-50 [1] as the feature extraction backbone of the network. We use the same setup as TESTR [6], the encoder contains a deformable transformer [7] with 6 layers, 8 heads, and 4 sampling points. Our encoder also contains a regular cross-attention layer with the queried regex, which also has 6 layers and 8 heads. Each one of the two decoder layers is composed of a deformable cross-attention layer between the features and the queries, the intra and inter-self attention layer of the queries, and the cross-attention layer between the queries and the encoded regex. All the decoder attention blocks have 6 layers and 8 heads, the deformable attention also uses 4 sampling points. The embedding dimension is 256 in all the cases. The number of queries of the decoders is 100.

The regex representation embedding is a feed-forward neural network that projects each one of the $\mathbf{h}_m$ vectors to the embedding size of the tokens. This layer is composed of 2 hidden fully connected layers of 256 dimensions and an output layer of the same size. Each layer has a ReLU activation function.

C.1. Model Loss

Our model follows the same training objectives described in [6]. The decoder training losses include, for each sub-query j, an instance classification loss $\mathcal{L}^j_{cls}$, an L1 distance loss $\mathcal{L}^j_{coord}$ for control-point regression, and a cross-entropy based character classification loss $\mathcal{L}^j_{char}$. Opposed to TESTR, the $\mathcal{L}^j_{cls}$ loss uses both the character and location

Figure 1. The target regular expression we want to find in Figure 1a is "[A-Za-z]{2}\d{2}[ ][A-Za-z]{2}\d{2}". Figure 1b shows the raw output of TESTR. The output instances are filtered by matching the two sub-queries resulting in splitting the main query by the space (in this case, "[A-Za-z]{2}\d{2}" for both sub-queries), shown in figure 1c. The two remaining instances are merged since their distance is lower than the established threshold. Figure 1d shows the final polygons and transcriptions merged into a single instance.

sub-queries to calculate the classification confidence, as opposed to location-only. The final decoder loss is $\mathcal{L}^j_{dec} = \sum_j(\lambda_{cls}\mathcal{L}^j_{cls} + \lambda_{coord}\mathcal{L}^j_{coord} + \lambda_{char}\mathcal{L}^j_{char})$. λ_{cls}, λ_{coord} and λ_{char} are used to weight the losses. We use the original values of $\lambda_{cls} = 2.0$, $\lambda_{coord} = 5.0$ and $\lambda_{char} = 4.0$. The proposals of the guidance generator of the encoder are also supervised with an instance classification loss $\mathcal{L}^j_{cls}$ and a coordinate regression loss $\mathcal{L}^j_{coord}$. Additionally, the encoder loss uses the generalized IoU loss $\mathcal{L}^j_{gIoU}$ defined by [5] for bounding box regression. The encoder loss is defined as $\mathcal{L}^i_{enc} = \sum_j(\lambda_{cls}\mathcal{L}^i_{cls} + \lambda_{coord}\mathcal{L}^i_{coord} + \lambda_{gIoU}\mathcal{L}^i_{gIoU})$, with $\lambda_{gIoU} = 2.0$.

D. Regex encoding

With the multi-hot encoding of the regex we can represent different matching operations. As described in section 3.1.1, the encoding $\mathbf{H}$ is composed by M multi-hot encoded vectors $\mathbf{H} = (\mathbf{h}_1, \ldots, \mathbf{h}_M)$. Each vector $\mathbf{h}_m$ has a total of K elements $\mathbf{h}_m = (h_{m,1}, \ldots, h_{m,K})$, where K is equal to the size of our character set. Each element k corresponds to a character of the set, and an element $h_{m,k}$ is set to 1 if that character matches the queried regex in that position. This representation provides a very fine-grained level of information at each position m but, while a flexible and powerful representation, it might be difficult to learn.

As an alternative to this approach, we have also tried a less complex encoding of the regex. Instead of using the multi-hot representation, we have tried a simpler and smaller one-hot approach. We redefine each vector $\mathbf{h}_m$ as $\mathbf{h}_m = (h_{m,1}, \ldots, h_{m,C})$, where C is equal to the number of classes we can represent at each character position m. We have defined a total of 6 classes; space, number, letter, separator (characters `,` `-` and `_`), special (rest of the alphabet), and padding. The disadvantage of this representation is that we no longer can match specific characters at a certain position. For example, the query "`A\d{2}0`" would not be possible.

Table 1 shows the End-To-End and Detection results on the evaluation set. The multi-hot approach obtains better re-

Figure 2. The target regular expression we want to find in Figure 2a is "\d{11}-\d". Figure 2b shows the raw output of TESTR. Figures 2c and 2d show two iterations of the instance merging process. In the first Figure, it has merged part of the UIC target code (the 12-number code) that was fragmented. The next step has fully merged the remaining bit of the code. Since the resulting transcription matches the query, this instance would not be further merged with others. Other instances that do not follow the query are removed, as seen in figure 2e.

sults in both the Detection and End-To-End tasks, although the average edit distance is slightly lower using the one-hot encoding.

E. Further Qualitative Analysis

E.1. Baselines Comparison

Figure 3 shows additional qualitative results of the baselines and STEP on samples from the test set. In the baseline results, we have not yet applied post-processing operations to showcase the raw output of these models. This Figure shows how the conditioning of our model results in a single detection, reducing the chance of detection failures and the need for post-processing operations.

E.2. Failure Cases

Figure 4 shows typical failure cases of STEP. Figure 4a shows a misspelled BIC code. While STEP's regex-based approach reduces spelling mistakes (such as mixing certain numbers and letters), it still can generate a wrong transcription among characters of the same type. Figure 4b shows a UIC failure case where the transcription has the wrong format (one of the digits is missing). UIC codes pose a particular challenge due to their length and multiple spaces. In figure 4c the model has wrongly detected and transcribed one of such cases. STEP is also not exempt from false positives, in Figure 4d the model has been given the regex of a UIC code and wrongly read unrelated text.

E.3. Multiple Instances

As Figure 5 shows, our model is capable of successfully detecting and recognizing multiple targets for a single query. During training, we match the generated regex with all the instances of the ground truth. Since multiple instances can match this query, our model learns that a single query can have multiple instances as the ground truth.

F. Test Dataset Details

Table 2 shows the number of instances for each one of the 6 codes and an example of their formats. Figure 6 also shows more examples of the different varieties of the codes of the test set.

References

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 1

[2] Mingxin Huang, Yuliang Liu, Zhenghao Peng, Chongyu Liu, Dahua Lin, Shenggao Zhu, Nicholas Yuan, Kai Ding, and Lianwen Jin. Swintextspotter: Scene text spotting via better

Encoding	End-To-End				Detection		
	Precision	Recall	F-score	Avg. ED	Precision	Recall	F-score
One-Hot	0.72	0.57	0.63	**0.11**	0.79	0.63	0.73
Multi-Hot	**0.78**	**0.64**	**0.71**	0.13	**0.86**	**0.69**	**0.76**

Table 1. End-To-End and Detection detections on the evaluation split using the One-Hot and Multi-Hot regex encoding approaches.

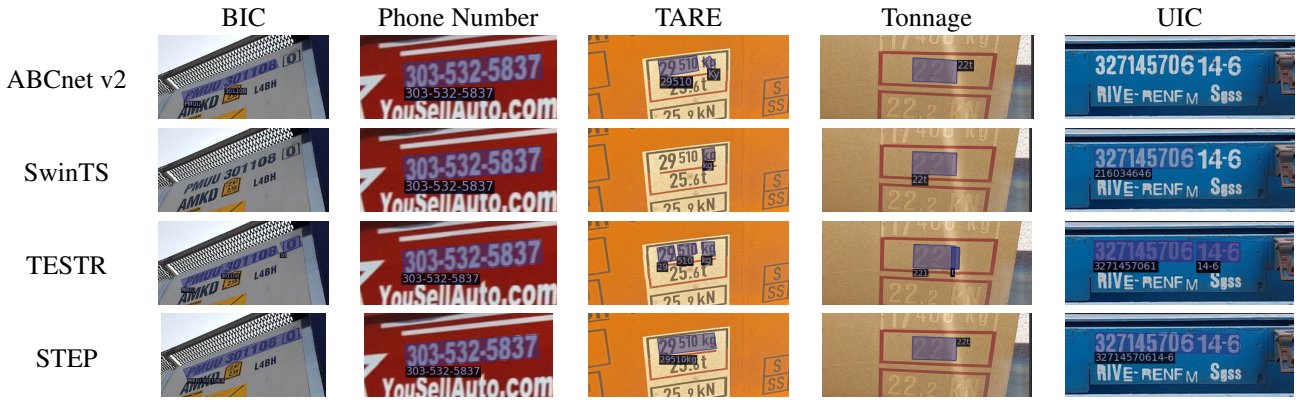


Figure 3. Qualitative results of the baselines and STEP on some examples from the test set.

Code Type	Num.	Example Format
BIC	329	"BICU 342894 0"
UIC	407	"2837 58 47 391-1"
TAREs	382	"25.000 KG" or "25000 kg"
Phone Numbers	109	"123-456-7890"
Tonnage	659	"25.0t" or "25t"
License Plates	121	"ABC 1234"

Table 2. Codes featured in our test split and examples of their format.

synergy between text detection and text recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4593–4603, 2022. 1

[3] Yuliang Liu, Hao Chen, Chunhua Shen, Tong He, Lianwen Jin, and Liangwei Wang. Abcnet: Real-time scene text spotting with adaptive bezier-curve network. In *proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9809–9818, 2020. 1

[4] Nibal Nayef, Fei Yin, Imen Bizid, Hyunsoo Choi, Yuan Feng, Dimosthenis Karatzas, Zhenbo Luo, Umapada Pal, Christophe Rigaud, Joseph Chazalon, et al. Icdar2017 robust reading challenge on multi-lingual scene text detection and script identification-rrc-mlt. In *2017 14th IAPR international conference on document analysis and recognition (ICDAR)*, volume 1, pages 1454–1459. IEEE, 2017. 1

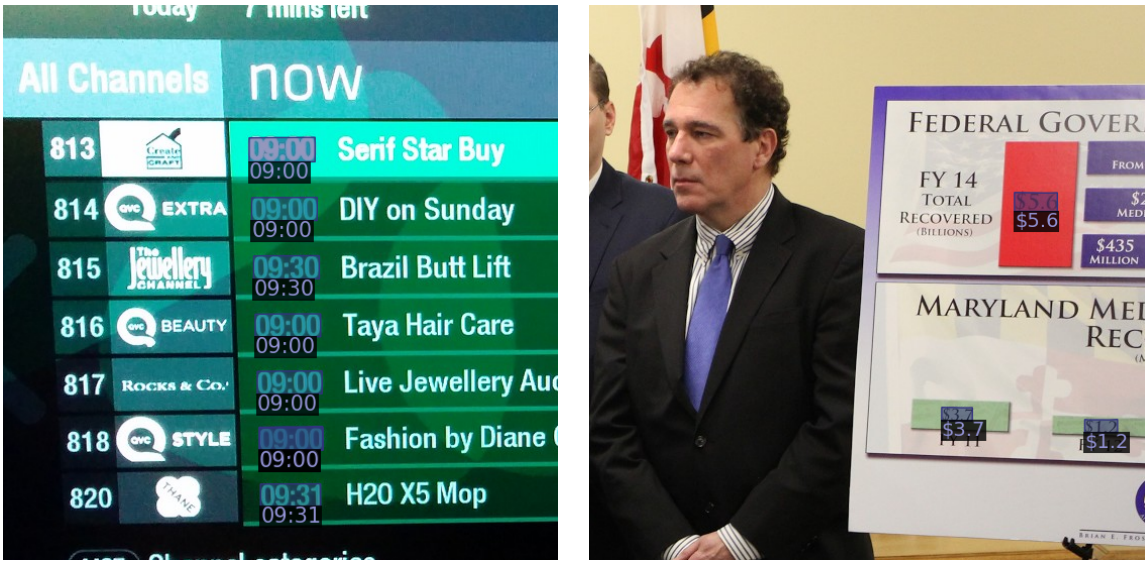
[5] Hamid Rezatofighi, Nathan Tsoi, JunYoung Gwak, Amir Sadeghian, Ian Reid, and Silvio Savarese. Generalized intersection over union: A metric and a loss for bounding box regression. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 658–666, 2019. 2

[6] Xiang Zhang, Yongwen Su, Subarna Tripathi, and Zhuowen Tu. Text spotting transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9519–9528, 2022. 1

[7] Xizhou Zhu, Weijie Su, Lewei Lu, Bin Li, Xiaogang Wang, and Jifeng Dai. Deformable detr: Deformable transformers for end-to-end object detection. *arXiv preprint arXiv:2010.04159*, 2020. 1

(a) (b) (c) (d)

Figure 4. Qualitative examples where STEP has failed to read a BIC code (4a), a UIC code (4b and 4c), and produced a false positive using the UIC regex (4d).



(a) Query: "\d{2}:\d{2}" (b) Query: "$\d.\d"

Figure 5. Our model is capable of detecting and reading multiple targets with a single query.

Figure 6. Examples of the different codes featured in our test dataset. The format of some codes, such as the UIC or BIC codes, feature different separations and spaces in their format.