Location Matters: Accelerating Historical Cipher Transcription with Detection-Based Models

George Lasry
The CrypTool Project
george.lasry@gmail.com

Abstract

Transcribing historical ciphers is the first step toward decryption and analysis. Machine learning models proposed for this task often neither consume nor produce symbol locations. Such location-discarding approaches do not integrate naturally with transcription and analysis tools that rely on a visual feedback loop linking image regions to transcribed symbols. We argue for location-aware, detection-based deep learning models that preserve location information in both supervision and output, supporting an end-to-end visual workflow with tools such as CTTS (CrypTool Transcription and Solver). To this end, we present two detection-based models with distinct architectures, evaluate them quantitatively across diverse cipher collections, and illustrate the workflow through a case study. The results show that this approach is practical and sample-efficient: it performs well with limited training data and remains effective in a challenging yet typical bootstrap scenario for new cipher collections. It supports human-in-the-loop correction, significantly reduces manual work, and helps produce accurate transcriptions and training data.

1 Introduction

The field of HTR (Handwritten Text Recognition) has been dominated in the last 15 years by segmentation-free approaches, in which models receive weak supervision in the form of line-level transcripts and, at inference, produce text sequences without character locations. This shift from prior segmentation-based approaches, which required explicit character boundaries, was made possible by the advent of CTC loss (Graves et al., 2006), bidirectional LSTMs, and later transformers (Li et al., 2023). This shift also affected HTR for historical documents, and most automated transcription solutions for texts from the Middle Ages and later — including leading platforms such as Transkribus (Muehlberger et al., 2019) and Kraken (Kiessling, 2019) — are built around segmentation-free architectures. Such approaches are especially effective for plaintext historical documents, where character-level annotation is costly, scripts are often cursive and connected, corpora are relatively large, alphabets are small and stable, and language priors compensate for weaker per-character supervision.

Ancient-script transcription projects differ. Not only are most ancient scripts non-cursive, with spatially separate symbols, but corpora are often limited, and the material poorly preserved, requiring every visual and spatial cue (2D, 3D, spectral imagery) to distinguish the features of each symbol. The ancient script alphabet itself may still be under investigation. As a result, most of those projects are built around detection-based architectures, such as DeepScribe for cuneiform (Williams et al., 2025) and YOLOv8 ensembles for Greek papyri (Turnbull and Mannix, 2025).

With historical handwritten, enciphered documents dating from the Renaissance to the 19th century, the picture is more nuanced, but still with a tendency to rely on segmentation-free approaches. This includes submissions to the ICDAR 2024 Competition on Historical Ciphers (Fornés et al., 2024), most of which do not produce symbol locations. Recent evaluations of HTR methods for ciphers (Souibgui et al., 2023) focus on such location-discarding approaches. Notable exceptions include few-shot detection (Souibgui et al., 2021), a detection-based model for cipher digits (Antal et al., 2022), and DTLR (Baena et al., 2024), a contest participant that is detection-based but trained partially without location annotations.

This paper makes three concrete contributions. First, it presents a detection-based approach to historical cipher transcription, validated on real-world collections totaling tens of thousands of symbols. Second, it introduces and evaluates two models with distinct architectures — a TIMM+FPN CenterNet-style detector and an adapted DINO-DETR/DTLR model — both

trained with explicit bounding-box supervision across diverse cipher collections. Third, it shows how tight integration into a transcription GUI (CrypTool Transcription and Solver – CTTS) enables a human-in-the-loop workflow that simultaneously produces high-quality transcriptions and new training data efficiently and at scale.

2 Why Symbol Locations Matter

In this article, we argue that transcribing documents with historical cipher symbols is much closer to transcribing ancient scripts than to transcribing plaintext handwritten documents. Several properties of ciphers undermine the advantages of segmentation-free approaches:

Unknown symbol sets. Cipher scripts use dozens to several hundred distinct types, adding a new layer of complexity to both transcription and interpretation. The symbol inventory itself is a research output, not an input — compound symbols, diacritical marks, and scribal variants are often discovered progressively.

Unknown keys. The cipher key, which maps specific symbols to language elements, is also unknown in most cases, adding one more layer of complexity. Transcription and decryption are coupled: errors in transcription often become apparent only through failed decryption attempts. This iterative cycle requires a spatial link between each symbol and its image location.

No language priors. Unlike plaintext, encrypted text offers no linguistic redundancy or clues. A segmentation-free model cannot – a priori – rely on a language model to resolve visual ambiguity, e.g., similarly-looking symbols, small diacritics, damaged or partially visible symbols, common in historical material.

Annotation cost. With plaintext in a known language, line-level transcription is fast because the annotator reads the text. With cipher symbols, the annotator inspects each unfamiliar symbol individually — effectively doing character-level work. As a result, transcribing a historical cipher document "as text", unless digits are employed, is no faster than annotating locations, which is not only convenient but also more accurate with a visual tool like CTTS.

Because of those additional layers of complexity, maintaining the link between transcripts and symbols is critical. The goal of processing historical cryptograms is to produce a fully decoded, accurate text ready for analysis. This requires a complex workflow — transcription, decryption, error correction, and analysis — in which errors introduced at one stage often become apparent only at another, and where all relevant elements should be integrated, viewed, and edited side by side without switching between applications. This includes the image with the symbols, the transcription, the nomenclature of symbol types (classes) and their meanings (the decryption key), the raw decrypted text, and the edited decryption in readable form. Such an integrated and interactive view is supported by tools such as CTTS. Location-aware models naturally support this iterative visual workflow.

Furthermore, while location-discarding models rely on "weak supervision", detection models benefit from "strong supervision" via explicit bounding boxes, and in most cases, can learn effectively from far fewer pages.

3 A Detection-Based Workflow

We propose a human-in-the-loop workflow that addresses the challenges of transcribing historical ciphers, for the typical case of an unknown symbol set and key:

1. **Bootstrap**: Manually transcribe a small initial set (~1,500 symbols) using CTTS, and train a first detection model.
2. **Assist**: Use the trained model to annotate the next batch, and review and correct the transcription with CTTS.
3. **Refine**: Retrain the model on the additional data, then repeat the process, improving both the data and the model.

Each cycle improves the model and expands the training set. Inference can even be run on training data to spot inconsistencies in earlier annotations, further improving dataset quality. The goal of producing high-quality transcriptions (for cryptanalysis, decryption, and analysis) and the goal of building quality training data become one and the same. In Section 11, we describe an alternative bootstrap approach using generic models.

4 Detection-Based Models – Early Investigations

We explored various options for a model based on existing components that would not require extensive computational power or lengthy training.[1] This included:

- A custom CNN with 5-channel output (objectness + offsets + size);
- YOLOv11 (yolo11x – 57M);
- ArcFace classification – two-stage pipeline: detect first, then classify cropped symbols (~4M params).[2]

5 First Model (TIMM)

We first implemented a model that combines well-established deep learning components into an end-to-end pipeline that jointly optimizes detection and classification objectives. The output consists of bounding boxes with class labels and confidence scores for each detected symbol, as well as a feature vector extracted from the fused FPN representation at the symbol's center location, which CTTS uses for clustering and outlier detection. The architecture (see Figure 1) consists of:

- **Backbone (via TIMM)**: Pretrained image classification networks—ConvNeXt (Liu et al., 2022) or ResNet (He et al., 2016) variants, with strong feature extraction without requiring massive training data. The script supports arbitrary TIMM backbones. We used one lighter configuration for fast-turnaround experiments,[3] and one heavier configuration for the final model used to transcribe documents.[4]
- **Feature Pyramid Network (FPN)** (Lin et al., 2017a): Top-down architecture with lateral connections that fuses high-resolution spatial detail with semantically rich deeper features. Essential for detecting symbols of varying sizes within the same line.
- **Detection heads** (CenterNet-style) Predict object centers as heatmap peaks rather than anchor boxes. Separate heads output center probability (with focal loss (Lin et al., 2017b)), bounding box size, and sub-pixel offset for precise localization.
- **Classification head**: Parallel branch predicting symbol class at each spatial location, trained jointly with detection using cross-entropy loss.

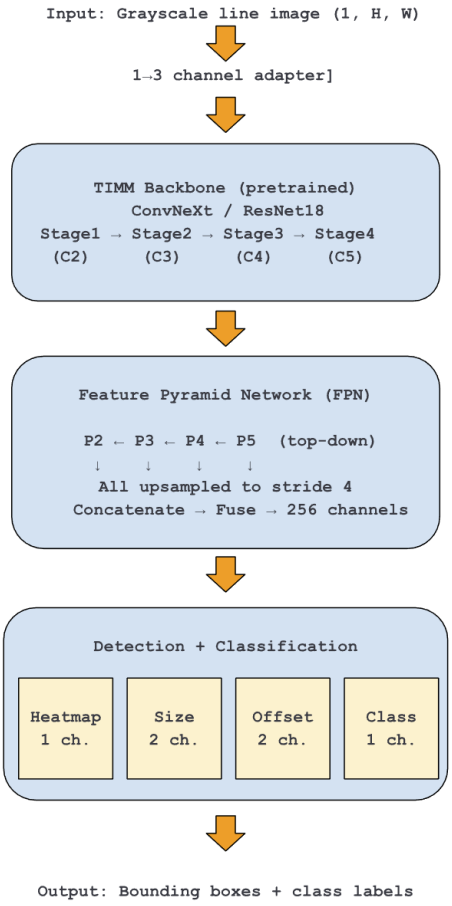

Figure 1. Model architecture with pretrained backbone, FPN decoder, and parallel detection/classification heads.

We trained using AdamW with a learning rate of 10^{-4} and weight decay of 10^{-4}, with cosine annealing. Input lines are cropped from page-level grayscale images, resized to 64 pixels

[1] With the assistance of LLM tools (Claude Code).

[2] Due to a lack of space, we do not reproduce here the results of those experiments. We discarded some options that could not produce embeddings or feature vectors needed by CTTS for clustering and outlier detection (see Section 9), like YOLOv11.

[3] *resnet18.fb_ssl_yfcc100m_ft_in1k*, ~11M parameters, pretrained self-supervised + ImageNet.

[4] *convnext_base.clip_laion2b_augreg_ft_in12k_in1k*, ~88M param., pretrained on ImageNet + CLIP. Reduces CER by a typical (relative) 20-30% compared to the lighter backbone (e.g., 3.5-4% vs. 5%)

height, width-capped, and padded to a fixed width for batching. We use focal loss (α=2.0, β=4.0) for the heatmap head. Training uses configurable probabilistic augmentation (including padding jitter, small rotations, brightness/contrast/gamma changes, blur, noise, and cutout), whereas validation is performed without augmentation. During inference, the default confidence threshold is 0.2, same-class NMS uses an IoU threshold of 0.3, cross-class NMS is disabled by default, and local peak detection uses a 3×3 kernel. Performance is evaluated using CER as the primary metric for model selection,[5] with precision, recall, and classification accuracy reported alongside. Training stops after 100 epochs or after 20 epochs without improvement in CER.

6 Case Study: The Borg Cipher

We illustrate the proposed workflow through a case study on the Borg cipher collection, a bootstrap scenario with no initial training data. This case is particularly challenging due to the irregular handwriting and intertwined symbols. Line segmentation was performed with CTTS and an algorithm based on horizontal projections.

6.1 Bootstrap – Manual transcription of an initial batch

We manually transcribed 5 pages using CTTS, totaling 1,519 symbols across 50 distinct classes.[6] This effort took 3.5 hours. This phase was also essential for understanding the cipher symbols' peculiarities, e.g., compound symbols.

6.2 First trained TIMM model

We trained the detection model on these initial data, using 40% of the material for validation.[7] Training was completed in 6 minutes. Results:

- Precision: 96.4%, Recall: 95.0%
- Classification accuracy: 96.1%
- CER: 11.02%

Figure 2 shows sample results on a test image: false negatives in red, false positives in orange, and misclassifications in purple. The relatively high segmentation recall and precision already make this early model useful and time-saving.

Figure 2. Sample results with the first model trained on Borg.

6.3 Batch with semi-automated transcription

Using this model, we processed a new batch of 5 pages. We manually fixed 37 false negatives, 2 false positives, and 38 misclassifications. Correcting the model's output took 45 minutes.

Figure 3 shows CTTS used to review and correct the transcription of a relevant page of the Borg cipher. All symbols of a given type can be viewed in one place, highlighting outliers.

[5] CER Character Error Rate is calculated based on the minimum number of character-level edits – substitutions (S), deletions (D), and insertions (I) required to transform the predicted text into the original reference text, divided by the total number of characters (N). CER = (S + D + I) / N. To qualify for a true positive, we employ loose IoU thresholds: 0.15 for the same class and 0.3 for a different class (misclassification).

[6] Only 50 of the 70 classes in the collection were observed in the first 5 pages.

[7] In this bootstrap scenario, we did not allocate holdout material for the final evaluation. The metrics are for the validation set.

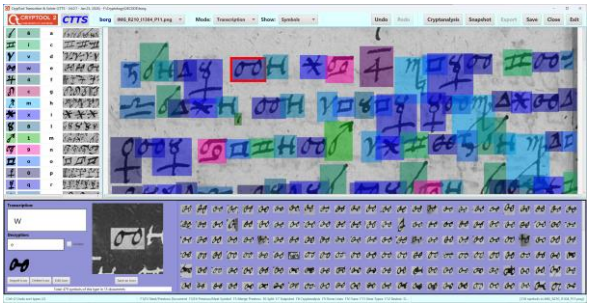

Figure 3. Working with CTTS to verify and improve the automated transcription.

6.4 Second trained TIMM model

We retrained the model on the extended dataset:

- Precision: 97.1%, Recall: 97.1%
- Classification accuracy: 96.9%
- CER: 7.06%

6.5 Additional batch with semi-automated transcription

With the improved model, we processed another 5 pages. Correction time dropped to 20 minutes, with 20 false negatives, a couple of false positives, and 35 misclassifications.

6.6 Third trained TIMM model

We retrained the model on the extended dataset:

- Precision: 98.3%, Recall: 97.4%
- Classification accuracy: 98.5%
- CER: 5.00%

7 Second Model – Adapted DINO-DETR/DTLR Model

In addition to the TIMM+FPN detector described in Section 5, we adapted the DTLR model of Baena et al. (2024) to be fine-tuned with explicit symbol bounding boxes and to predict both the class and position of detected symbols.

DTLR is a text line recognition model built on the DINO-DETR object detection architecture (Zhang et al., 2022), trained in two stages: pretraining on synthetic data with full detection supervision, then fine-tuning on real data with weaker sequence-level supervision. A ResNet-50 backbone extracts multi-scale feature maps from a cropped line image, which are refined by a transformer encoder (6 layers of multi-scale deformable self-attention) and decoded by a transformer decoder (6 layers). The decoder uses 900 learned content queries, each paired with a positional anchor initialized from the encoder output; after self-attention and deformable cross-attention to the encoder features, each query predicts a 4-coordinate bounding box and a probability vector over the symbol alphabet. During pretraining, the Hungarian algorithm matches predictions to ground truth, and the loss combines focal loss for classification with L1 and generalized IoU losses for box regression. The model produces all detections for a line in parallel, unlike autoregressive approaches that generate one symbol at a time. In the original DTLR pipeline (Figure 4, left side), this architecture is pretrained on synthetic line images of Latin characters with exact positions and identities. The resulting pretrained "Latin" model is then fine-tuned on real ciphertext data using only line images and transcribed symbol sequences, without bounding boxes, via an adapted CTC loss: the predicted boxes order the character queries spatially, and CTC aligns this ordered sequence with the ground-truth transcription. At inference, the fine-tuned DTLR model primarily outputs an ordered sequence of symbols.

Our adaptation (Figure 4, right side) consumes and produces symbol locations. We start from the same pretrained Latin DINO-DETR model released by Baena et al., but replace the classification heads with new layers sized to the target cipher's symbol inventory. Their rows are initialized by copying weights and biases from randomly selected rows of the pretrained Latin classifier, rather than from a fully random initialization, following Baena et al.'s transfer strategy. Instead of fine-tuning with CTC loss on transcribed sequences, we fine-tune with explicit symbol-class and bounding-box supervision, using the same detection loss as in DTLR pretraining: focal loss for classification and L1 plus GIoU losses for box regression. At inference, the adapted model outputs both symbol classes and bounding boxes.

There are two additional practical differences. First, DTLR fine-tunes on binarized line crops, whereas we use grayscale line crops extracted from manuscript pages. Second, our implementation splits long lines into overlapping segments for both training and inference to accommodate the variable, often extended line lengths typical of historical ciphers; the original DTLR processes each line as a single input.

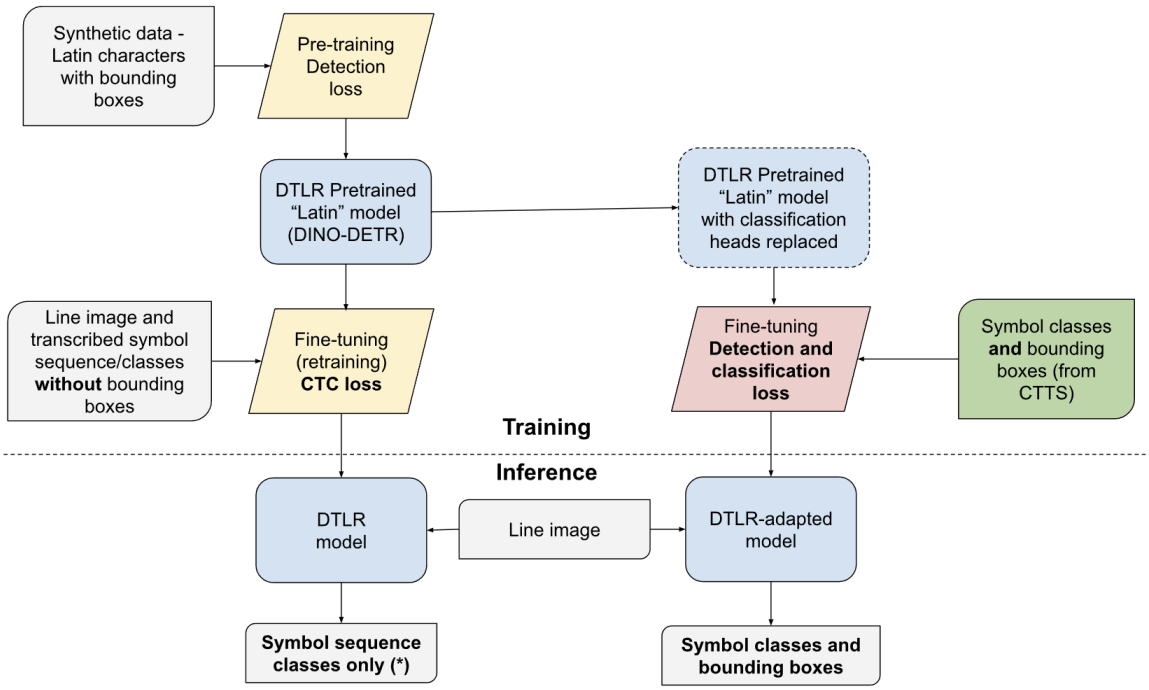

Figure 4. Comparison of the original DTLR pipeline (left) and our adaptation (right). Both start from the same pretrained Latin DINO-DETR/DTLR model. The original pipeline fine-tunes with CTC loss using only transcribed sequences without bounding boxes, producing primarily ordered class sequences at inference. Our adaptation fine-tunes with the full detection-and-classification loss, producing both symbol identities and locations at inference.

8 Performance Evaluation

Training sessions on collections of encrypted documents, transcribed with CTTS, were conducted using an NVIDIA RTX 3090 and took between 6 and 25 minutes for the TIMM model, and up to 90 minutes for the DINO model. We trained models on increasing amounts of training materials, pre-allocating a fixed holdout set of 20,000 symbols (10,000 for Borg) for evaluation. For validation and selecting the best epoch, we used 20% of the training material. As shown in Table 1, the models performed well across multiple collections of ciphertexts, including a subset of the recently deciphered letters from Mary Stuart (Lasry et al., 2023), comprising 157 distinct classes.[8]

Four of the collections were included in the ICDAR contest (Copiale, Borg, Ramanacoil, and BnF). The CER figures are not directly comparable to those of the contest, because our models were trained with strong supervision on grayscale images with explicit symbol boxes and class labels, whereas the contest systems used weaker supervision on binarized segmented lines with transcripts only.[9] We do include the ICDAR data in Table 1 because they provide useful context on training scale: the contest used 2,594 training lines for Borg, 1,502 for Copiale,[10] 1,291 for Ramanacoil, and 788 for BnF, whereas our evaluation emphasizes bootstrap settings with much smaller training sets under strong supervision.[11]

[8] A similar model is currently being used to transcribe another collection of ciphertexts related to Mary Queen of Scots, with more than 100,000 symbols.

[9] Our models were trained on datasets generated using CTTS, which we curated and refined.

[10] The Copiale's thin vertical symbols (bars, iotas) were a significant source of errors in earlier versions of the TIMM model. Targeted augmentation strategies substantially reduced these errors in later versions.

[11] The results on the BnF dataset stand out as a case where strong supervision does not provide a substantial sample-efficiency advantage. The BnF dataset has a small symbol set (36) and clean, well-separated symbols.

Dataset	Training Lines	Training Symbols	Number of Classes	Average Sym./Class	Model	CER (%)	Prec. (%)	Recall (%)	Class. (%)
Borg	165	2500	70	35.7	TIMM	6.6	98.6	98.5	96
					DINO	6.75	98.9	97.9	96.2
Borg	311	5000	70	71.4	TIMM	5.55	98.9	98.7	96.7
					DINO	5.54	99.1	97.7	97.4
Borg	640	10000	70	142.8	TIMM	4.26	99	99	97.6
					DINO	4.96	98.6	98.6	97.7
In the ICDAR contest, the best CER on the Borg dataset was 6.76% with **2594** training lines and weak supervision (sequence-only)									
Copiale	67	2500	113	22.1	TIMM	10.97	99.3	98.3	91
					DINO	4.46	99.7	98.1	97.6
Copiale	119	5000	113	44.2	TIMM	4.12	99.7	98.9	97.2
					DINO	2.57	99.7	99.1	98.6
Copiale	239	10000	113	88.5	TIMM	2.37	99.7	99.3	98.6
					DINO	1.61	99.8	99.4	99.2
Copiale	480	20000	113	177	TIMM	1.32	99.9	99.5	99.3
					DINO	1.21	99.7	99.6	99.4
In the ICDAR contest, the best CER on the Copiale dataset was 1.62% with **1502** lines for training and weak supervision									
Ramanacoil	68	2828	71	39.8	TIMM	6.84	98.7	98.4	95.6
					DINO	16.16	97.9	91.7	93.1
Ramanacoil	84	4990	71	70.3	TIMM	3.18	99.1	99.1	98.2
					DINO	6.37	98.5	97.1	97.7
Ramanacoil	201	10000	71	140.8	TIMM	1.85	99.7	99.4	99
					DINO	4.99	98.6	98	98.2
In the ICDAR contest, the best CER on the Ramanacoil dataset was 5.61% with **1291** training lines with weak supervision									
BnF Fr. 3029	323	10000	36	277.8	TIMM	1.21	99.8	99.8	99.1
					DINO	1.73	99.6	99.7	98.9
In the ICDAR contest, the best CER for the BnF Fr., 3029 dataset was 0.89% with **788** lines for training and weak supervision									
Mary Stuart	270	20004	157	127.4	TIMM	2.46	99.9	99.7	97.9
					DINO	4.13	98.8	99.2	97.8

Table 1. Evaluation results across cipher collections with our two detection-based models trained with strong supervision. ICDAR 2024 data are included as contextual reference points for weak-supervision settings, to illustrate the amount of training data used in the contest. The CER figures are not directly comparable.

Table 1 also shows that the two architectures have complementary strengths. In these experiments, TIMM is generally more robust in low-data bootstrap settings and performs better on Borg, Ramanacoil,[12] and BnF, whereas the adapted DINO model becomes competitive or superior on some datasets, especially Copiale. In practice, this makes TIMM a strong default model, while DINO provides a useful alternative whose strengths may depend on the symbol inventory and visual characteristics of the documents in a collection. As described in Section 12, the availability of a second model with a different architecture has several additional advantages, e.g., enabling an independent review of the results of the first model.

We analyzed the results of 80 training runs with the TIMM model, with datasets from various origins, sizes, and types, transcribed with CTTS. We found a strong correlation between the best CER achieved and the ratio of the total number of symbols used for training to the number of distinct symbol classes (the average number of times a given class appears in the training data), as shown in Figure 5.[13] Recall and precision reach 90%+ in most cases with fewer than 1000 symbols, as shown in Figure 6.

9 Integrating the Models into CTTS and Transcription Workflows

To integrate the models into CTTS, we use ONNX (Open Neural Network Exchange), an open format for representing trained machine learning models, allowing them to be exported from one framework (e.g., PyTorch) and run in another runtime environment without requiring the original training code. An ONNX model stores both the model's architecture (as a computation graph) and its learned weights in a single portable file. CTTS with an ONNX model loaded can detect symbols at a rate of 5 to 50 symbols per second without a GPU, depending on the number of CPU cores and the model (detection with TIMM models is faster). A GPU can accelerate detection by a factor of 5-10.

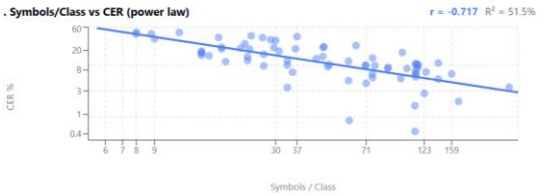

Figure 5. TIMM model – CER as a function of (number of symbols/number of classes).

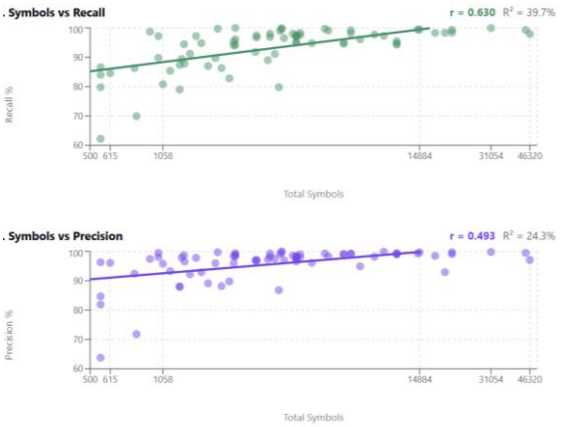

Figure 6. TIMM model – precision and recall as a function of training data size.

In addition to the location and identity of the symbols in a document, the deep learning models also produce symbol detection features (TIMM models), or symbol embeddings (DINO models), which are used by CTTS for clustering similar symbols into tentative classes, auto-classifying newly detected symbols by comparing their features/embeddings with those of previously transcribed symbols, or identifying outliers and detecting potential classification errors.[14]

10 Towards a Universal Detection Model for Cipher Symbols

So far, we have described a bootstrap workflow in which some material is manually transcribed, and an initial model is trained on it; additional pages are transcribed with this model and reviewed manually, and so on. We are also evaluating

[12] The source of DINO's weaker performance on Ramanacoil is not fully understood; the collection's exceptionally long lines, not consistently straight, may play a role, although our tiling strategy is designed to handle such cases.

[13] Some of the correlations can be partially explained by symbols not seen or rarely seen in training data.

[14] Such detection features or embeddings are not produced by non-detection-based models, obviously. Preliminary results indicate that the embeddings produced by our DINO models are more discriminative than those from TIMM models. We are also experimenting with a Real-Time DETR architecture, which is much faster while still producing high-quality embeddings.

generic models trained on a variety of historical cipher documents and symbols (~400 classes). Those models yield locations with high precision and recall (90-95%) with ciphertexts with symbol types unseen during training. Classification is partially useful as symbols of the same type tend to be assigned to the same class, and similarly, symbols assigned to a given class tend to be similar. Generic models can accelerate the end-to-end transcription process compared to manual transcription, and are being integrated into CTTS.

11 An Alternative Bootstrap Workflow

With the tight integration of generic models with CTTS, we can propose an alternative bootstrap scenario that requires even less manual work, no model training or retraining, and no GPU resources, all the steps performed with CTTS:

- Apply a **generic model** to automatically transcribe a few initial documents.
- Apply the **CTTS clustering** function to improve the initial classifications.
- Review and manually correct this initial transcription and classification, leveraging **CTTS's built-in tools to detect outliers**.
- Apply the generic model again to additional documents.
- **Auto-classify the symbols with CTTS** (which compares their embeddings to those of already transcribed symbols).
- Review the transcription and classifications and repeat those steps. CTTS autoclassification becomes more accurate as more data is transcribed.

12 Working with Multiple Models

If we can train two fine-tuned models on the same symbol set, it is possible (with CTTS) to apply one model (e.g., TIMM) to detect and classify symbols in documents, and then apply the second model (e.g., DINO) to generate a list of edit suggestions. This feature is powerful, enabling high-quality transcriptions with minimal effort. Since the TIMM and DINO architectures described here differ, they often produce different types of errors, and the probability that both will make the same error is relatively low.[15]

13 Conclusion

In this article, we highlighted the advantages of working with detection-based models. First, by outputting bounding boxes alongside class predictions, deep learning models integrate directly into tools like CTTS, maintaining the visual feedback loop required for efficient correction and ultimately producing rigorous, publication-quality transcriptions and decryptions. Second, precise supervision via explicit bounding boxes enables models with diverse architectures to generalize from training sets of limited size, which is the general case with historical ciphers.[16] Third, detection models achieve high segmentation precision and recall early, often before they can achieve high classification accuracy. This allows collections to be processed incrementally, with early versions already relieving the transcriber of the most laborious task — drawing thousands of bounding boxes from scratch—and leaving only the faster work of verification and relabeling.

Our case study on the Borg cipher demonstrates these advantages across an entire bootstrap workflow, where initially, no training data was available. Current work on generic models for detecting cipher symbols will enable such bootstrap scenarios to be implemented at even lower cost (in terms of manual effort and GPU resources). Overall, this approach, together with tight integration with CTTS, significantly accelerates transcription, including large-scale collections, and improves its quality. Higher quality transcriptions enable more accurate decryption and key recovery, which are essential for historical research. This approach also improves the quality of the training data.

We are planning to release the scripts, the models, and the integrated version of CTTS.

[15] With this approach, we have transcribed hundreds of thousands of symbols from multiple collections, including large ones, with minimal effort. We are also evaluating the option of producing ensemble predictions from several models.

[16] The two models we presented already leverage pretrained models. Further research may include vision foundation models such as SAM2 for segmentation, or DINOv2 for feature extraction, pretraining on synthetic data consisting of cipher symbols, pseudo-labeling, and other detection and classification architectures. In case the cipher key is known, language models may also help improve classification accuracy.

Acknowledgments

We would like to thank Raphael Baena and his co-authors for publishing their code and models and making them available. We also thank the anonymous reviewers for their valuable feedback.

References

Eugen Antal and Pavol Marák. 2022. Automated Transcription of Historical Encrypted Manuscripts. *Tatra Mountains Mathematical Publications,* 82(2): 65–86.

Raphael Baena, Syrine Kalleli, and Mathieu Aubry. 2024. General Detection-Based Text Line Recognition. In *NeurIPS 2024*.

George Lasry. 2026. *CTTS – CrypTool Transcriber & Solver*. In *Proceedings of the 7th International Conference on Historical Cryptology (HistoCrypt 2026)*, Linköping University Electronic Press.

Alicia Fornés, Jialuo Chen, Pau Torras, Carles Badal, Beáta Megyesi, Michelle Waldispühl, Nils Kopal, and George Lasry. 2024. ICDAR 2024 Competition on Handwriting Recognition of Historical Ciphers. In *ICDAR 2024*. Springer.

Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. 2006. Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks. In *ICML 2006*.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *CVPR 2016.*

Benjamin Kiessling. 2019. Kraken – a Universal Text Recognizer for the Humanities. *DH 2019, Book of Abstracts.*

George Lasry, Norbert Biermann, and Satoshi Tomokiyo. 2023. Deciphering Mary Stuart's Lost Letters from 1578–1584. *Cryptologia,* 47(2): 101–202.

Minghao Li, Tengchao Lv, Jingye Chen, Lei Cui, Yijuan Lu, Dinei Florencio, Cha Zhang, Zhoujun Li, and Furu Wei. 2023. TrOCR: Transformer Based Optical Character Recognition with Pretrained Models. In *AAAI 2023*.

Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. 2017a. Feature Pyramid Networks for Object Detection. In *CVPR 2017*.

Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. 2017b. Focal Loss for Dense Object Detection. In *ICCV 2017*.

Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. 2022. A ConvNet for the 2020s. In *CVPR 2022.*

Guenter Muehlberger, Louise Seaward, Melissa Terras, Sofia Ares Oliveira, Vicente Bosch, Maximilian Bryan, Sebastian Colber, et al. 2019. Transforming Scholarship in the Archives Through Handwritten Text Recognition: Transkribus as a Case Study. *Journal of Documentation,* 75(5): 954–976.

Mohamed Ali Souibgui, Alicia Fornés, Yousri Kessentini, and Crina Tudor. 2021. A Few-Shot Learning Approach for Historical Ciphered Manuscript Recognition. In *ICPR 2020*.

Mohamed Ali Souibgui, Pau Torras, Jialuo Chen, and Alicia Fornés. 2023. An Evaluation of Handwritten Text Recognition Methods for Historical Ciphered Manuscripts. In *HIP '23*. ACM.

Robert Turnbull and Evelyn Mannix. 2025. Detecting and Recognizing Characters in Greek Papyri with YOLOv8, DeiT and SimCLR. *International Journal on Document Analysis and Recognition (IJDAR),* 28: 277–285.

Kathryn Williams, Yingjie Su, David Schloen, Sandra Schloen, Miller Prosser, Susanne Paulus, and Sanjay Krishnan. 2025. Localization and Classification of Elamite Cuneiform Signs on Persepolis Fortification Tablets. *Journal on Computing and Cultural Heritage,* 18(1): 1–22.

Hao Zhang, Feng Li, Shilong Liu, Lei Zhang, Hang Su, Jun Zhu, Lionel M. Ni, and Heung-Yeung Shum. 2022. DINO: DETR with Improved DeNoising Anchor Boxes for End-to-End Object Detection. In *ICLR 2022*