



Master's thesis
Master's Programme in Data Science

Identifier Extraction from Engineering Drawings via Object Detection and OCR

Mia Lindblad

May 5, 2026

UNIVERSITY OF HELSINKI
FACULTY OF SCIENCE
P. O. Box 68 (Pietari Kalmin katu 5)
00014 University of Helsinki

Tiedekunta — Fakultet — Faculty		Koulutusohjelma — Utbildningsprogram — Degree programme	
Faculty of Science		Master's Programme in Data Science	
Tekijä — Författare — Author			
Mia Lindblad			
Työn nimi — Arbetets titel — Title			
Identifier Extraction from Engineering Drawings via Object Detection and OCR			
Työn laji — Arbetets art — Level		Aika — Datum — Month and year	
Master's thesis		May 5, 2026	
		Sivumäärä — Sidantal — Number of pages	
		49	
Tiivistelmä — Referat — Abstract			
Engineering drawings include critical sensor and cable identifiers that frequently exist only as raster graphics within PDF documents, making lookups and validations slow and error-prone. This thesis investigates whether sensor and cable information can be automatically extracted from electrical engineering drawings by combining object detection and Optical Character Recognition (OCR) to produce structured identifiers reusable in downstream systems. The proposed solution is an end-to-end pipeline developed for internal Wärtsilä drawings. A YOLOv5m-based object detection model localises small sensor and cable symbols on high-resolution pages, and post-processing applies class-specific confidence thresholds and non-maximum suppression. A data sufficiency analysis examines how additional annotations impact class-wise precision and recall. For text extraction, several OCR engines are benchmarked on short alphanumeric IDs cropped from detected symbols: PyTesseract, Amazon Rekognition, and a multimodal GPT-based model. In addition, automatic bounding-box correction strategies are analysed. On a held-out test set, the final detector achieves a precision of 0.99 and a recall of 1.00 for both cables and sensors after post-processing, and Amazon Rekognition recovers approximately 97–98% of sensor and cable IDs from detector crops. The integrated pipeline stores extracted classes and IDs in a database table for search and traceability within internal tools. This work is limited to data from a single company and a small test set with two imbalanced symbol classes and relies on one detector family and a single commercial OCR service without custom post-correction. As such, reported performance is specific to this context, and transferring the pipeline to other contexts requires additional data and validation. Results show that automatic extraction of sensor and cable IDs from engineering drawings is feasible with current tools and can reduce manual effort when used as an assistive system. ACM Computing Classification System (CCS): Applied computing → Document management and text processing → Document capture → Optical character recognition Computing methodologies → Artificial intelligence → Computer vision → Computer vision problems → Object detection			
Avainsanat — Nyckelord — Keywords			
engineering drawings, object detection, optical character recognition, information extraction			
Säilytyspaikka — Förvaringsställe — Where deposited			
Muita tietoja — Övriga uppgifter — Additional information			

Contents

Acknowledgements	1
1 Introduction	3
2 Background and Related Work	5
2.1 Modern Approaches for Symbol and Text Extraction	5
3 Research Questions	9
4 Methods	11
4.1 Data Collection and Labelling	12
4.2 Tiling for Small Object Detection	13
4.3 Evaluation Metrics	14
4.4 Post-Processing	14
4.5 Data Sufficiency Analysis	15
4.6 Baseline Model	17
5 Hyperparameter Optimisation	19
5.1 Motivation for Hyperparameter Tuning	19
5.2 Hyperparameter Search Strategy	19
5.3 Search Space	20
6 Optical Character Recognition	23
6.1 OCR Problem Definition and Text Fields	23
6.2 Cropping and Detector-Based Preprocessing	24
6.3 OCR engines	24
6.3.1 PyTesseract Configuration	25
6.3.2 AWS Rekognition Configuration	25
6.3.3 GPT-5.0 Configuration	25
6.4 Shared Evaluation Protocol and Metrics	26

7	Results	27
7.1	Object Detection Results	27
7.1.1	Experimental Setup	27
7.1.2	Baseline Model Performance	28
7.1.3	Hyperparameter Optimisation Results	30
7.2	OCR Results	32
7.2.1	PyTesseract Results	32
7.2.2	AWS Rekognition Results	33
7.2.3	GPT-5.0 Results	34
7.2.4	OCR Engine Comparison	34
7.3	Automatic Bounding Box Quality Assessment for OCR Optimisation	34
7.3.1	Motivation and Problem Statement	35
7.3.2	Investigated Approaches: Padding	35
7.3.3	Investigated Approaches: Database	37
7.3.4	Automatic Bounding Box Quality Assessment Conclusions	38
7.4	Final Pipeline Results	38
7.4.1	Evaluation Setup	38
7.4.2	Detection Component Summary	39
7.4.3	OCR Component Summary	39
7.4.4	End-to-End ID Extraction Accuracy	40
7.4.5	Implications for Deployment	40
8	Discussion	41
9	Conclusions	45
	Bibliography	47

Acknowledgements

Commissioned by Wärtsilä Oyj Abp, this thesis investigates automatic extraction of sensor and cable identifiers from engineering drawings to support more efficient engineering workflows at the company.

I would like to thank my thesis supervisor at the University of Helsinki, Giulio Jacucci, for his valuable feedback and dedication in reviewing my thesis. I also wish to express my appreciation to my manager at Wärtsilä, Tal Katzav, for initiating and supporting this project. I am deeply grateful to my primary thesis supervisor at Wärtsilä, Maria Litova, for her helpful discussions, constructive feedback, and timely encouragement that facilitated progress. My gratitude extends to Tony Savander for his business-oriented support and to my friends, family, and all others who have supported me during this journey.

1. Introduction

Engineering drawings contain structured technical data, primarily available as PDFs or images with graphical symbols and alphanumeric labels. Assemblers responsible for system integration regularly consult multiple diagrams to locate sensor and cable identifiers and to trace connections throughout the system. This manual search is inefficient and prone to human error. The motivation for this thesis is to improve the accuracy and consistency of these search tasks. If information from engineering drawings can be digitally extracted and stored in a structured form, it becomes available to internal software tools for search, traceability, and label verification, for example by quickly locating all occurrences of a given identifier or flagging missing or inconsistent IDs across diagrams.

This thesis presents an approach for automated identification and extraction of sensor and cable data from engineering drawings. The scope is restricted to the detection of sensor and cable symbols and the extraction of associated or embedded text. Due to the proprietary nature, density and high resolution of the drawings, conventional models require retraining on domain-specific datasets. The methodology involves training an object detection model to identify sensors and cables, using bounding boxes to extract image crops for Optical Character Recognition (OCR), and evaluating multiple OCR methods for reading concise alphanumeric IDs. The objective is an end-to-end pipeline that outputs structured sensor and cable identifiers for downstream integration.

In the Wärtsilä use case, the extracted sensor and cable IDs are stored in a database, where internal tools can query them to support search across drawings and help engineers trace components without manually scanning multiple diagrams. The contribution of this thesis is therefore not a new detector or OCR architecture, but a task-specific evaluation of a complete extraction pipeline for short engineering identifiers. Compared with prior pipelines that often target broader drawing understanding or report detection and OCR stages separately, this work studies how annotation volume affects class-wise small-symbol detection, compares OCR engines on detector-generated crops of short alphanumeric IDs, and quantifies how bounding-box truncation and post-hoc correction attempts affect end-to-end ID-level accuracy.

Chapter 2 reviews past and current approaches for extracting structured information from technical documents and engineering drawings. Chapter 3 lists the specific research questions addressed in this thesis. Chapter 4 describes the object detection pipeline. It covers data collection and labelling, tiling, post-processing, evaluation metrics, and a data sufficiency analysis. Chapter 5 explains how model hyperparameters were tuned and why this was needed to improve detection performance. Chapter 6 focuses on the OCR component. It compares three OCR methods and their pre-processing requirements. Chapter 7 presents the evaluation of both the baseline and optimised object detection models, the OCR experiments, including the proposed correction strategies and their results, and the full end-to-end performance on Wärtsilä drawings.

2. Background and Related Work

The challenge of extracting structured information from technical documents predates the advent of present-day computing [22]. In particular, technical drawings and scanned documents are difficult because they mix textual and graphical elements. This was a major obstacle for researchers even before digital tools were available [22].

Early systems focused on separating text from graphics. In 1988, one algorithm split text strings from mixed content. However, the full algorithm took 764.31 seconds per 2048×2048 pixel image [9]. By 1998, researchers used the structure found in drawings. One system extracted dimension text by dividing text boxes according to ISO or ANSI standards. It reached 89% accuracy in both recognition and extraction [6]. Other studies relied on known drawing elements. A 2001 logic-based method found patterns using fixed pixel sizes for symbols, lines, and arrows [4]. A 2007 system added human feedback loops to improve symbol recognition [29].

2.1 Modern Approaches for Symbol and Text Extraction

Deep learning has substantially changed how symbol detection in engineering drawings is carried out. Convolutional Neural Network (CNN)-based detectors are commonly used for both localisation and recognition. A 2018 paper tested CNNs for this task but noted a lack of standard benchmark datasets as a fundamental bottleneck [7].

Two years later, researchers applied deep learning to Piping and Instrumentation Diagrams (P&IDs). They paired a YOLO detection model with General Adversarial Networks (GANs) for classification. Synthetic data helped balance rare classes [8]. In 2021, another team digitised large technical drawings using Faster R-CNN for object detection and character-based models for text, reporting an average text recognition accuracy of 81.8% on their benchmark [23].

In 2023, some clearer trends emerged. One review mapped neural network methods for engineering drawings and found continuing issues, such as scarce datasets and complex layouts, which lead to false positives or missed detections [21]. The review

called YOLOv7 promising. Another 2023 paper used multiple YOLOv7 models for 2D drawings. These models handled view segmentation, annotation types such as dimensions and Datum identifiers, and Geometric Dimensioning and Tolerancing (GD&T) symbols, with OpenCV preprocessing along with Tesseract OCR [15, 26]. Riku Helle’s 2023 master’s thesis gives a practical example. Riku extracted GD&T symbols from engineering drawings to estimate costs based on drawing complexity. Furthermore, Riku built datasets with Label Studio, detected text using PyTesseract, and compared YOLOv7 with Faster R-CNN [11].

Recent pipelines that chain detection, OCR, and post-processing are now more systematic. A 2025 paper introduced the eDOCr2 framework for mechanical drawings via Intelligent Document Processing (IDP), combining OpenCV, PyTesseract, custom CRNNs, and multimodal models in specialised pipelines for tables, feature control frames, and dimensions. The Table pipeline merged PyTesseract with Qwen2-VL-7B, while dimensions used CRAFT-based text detection and several OCR engines; ablation studies showed that sliding-window segmentation, template matching, and stroke correction substantially improved recall and character error rate on benchmark drawings [27].

Another 2025 effort built a hybrid vision-language system for 2D drawings: YOLOv11-obb detection fed into Donut [14] or Florence-2 [30] transformers for text parsing. Fine-tuned on 1367 annotated drawings, it compared parser options, but struggled with hallucinations [13].

Nuclear engineering offers a related case. One team automated equipment tag detection to reduce manual effort and errors that affect safety and efficiency. They combined Azure OCR (high precision) with Azure Document Intelligence (wider coverage, lower accuracy), merging results for gains. Preprocessing methods such as morphological filtering, colour inversion, and 3×3 grid tiling improved small-element recognition. GPT-4 post-processing helped validate outputs, though similar-character confusion persisted [10].

Symbol and text recognition in engineering diagrams remains challenging, as the drawings are densely packed with detail and are usually proprietary, requiring internal datasets and hindering benchmarking. Existing methods range from classical OCR tools (such as Tesseract and Azure OCR) to learning-based and multimodal models, and several works now chain detection, OCR, and post-processing into pipelines. The nuclear engineering paper, although in a different domain, demonstrates a useful strategy for small objects: tiling large drawings into smaller patches, improves recognition performance for dense, cluttered tags [10]. A similar tiling approach is adopted in this thesis to preserve small sensor and cable symbols on high-resolution pages.

Based on the studies reviewed in this chapter, three gaps appear particularly

relevant to the present thesis. First, few of the reviewed studies report a focused comparison of OCR engines on short alphanumeric identifiers extracted from engineering drawings. Second, detector bounding-box quality is rarely analysed in terms of its downstream effect on end-to-end identifier extraction accuracy. Third, the reviewed literature provides limited discussion of post-hoc strategies for handling truncated detector crops or validating OCR outputs when no single regular expression describes the identifier format.

3. Research Questions

This thesis investigates automatic information extraction from engineering drawings using object detection and OCR. In line with the gaps identified in Chapter 2, the focus is on four aspects: end-to-end extraction of sensor and cable IDs into a reusable table for internal search and lookup, the effect of annotation volume on detector behaviour, the performance of different OCR engines on short technical identifiers, and the downstream impact of detector crop quality on OCR accuracy.

The research questions are:

1. (RQ1) How accurately can an object detection pipeline localise sensor and cable symbols in engineering drawings?
2. (RQ2) How does the amount of annotated training data affect class-wise detection performance for sensor and cable symbols?
3. (RQ3) Given detector-generated crops, which OCR engine achieves the highest exact-match accuracy for short alphanumeric identifiers?
4. (RQ4) To what extent do bounding-box truncation errors affect end-to-end identifier extraction, and can simple post-hoc correction strategies improve the final ID accuracy?

4. Methods

The proposed pipeline consists of five stages: PDF-to-image conversion, manual annotation, object detection, bounding-box post-processing, and OCR-based ID extraction. Azure Machine Learning Studio [17] was used for annotation and detector development, while the downstream OCR experiments were implemented as a separate evaluation pipeline. Figure 4.1 provides an overview of the full workflow.

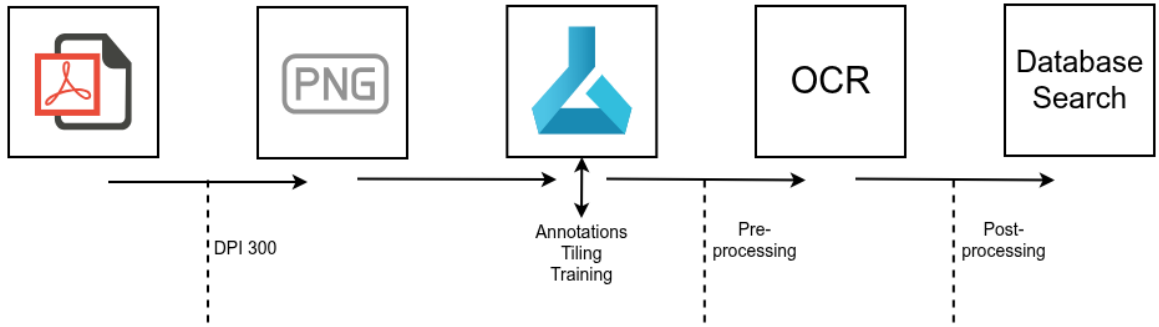


Figure 4.1: Overview of the end-to-end pipeline used in this thesis. Raw engineering drawing PDFs are converted to PNG images, which are then used in Azure Machine Learning Studio for annotation, pre-processing steps, and training an object detection model. The resulting bounding boxes are post-processed before being forwarded to the OCR engine. Finally, post-OCR cleaning and possible ID validation steps produce the structured sensor and cable identifiers that are stored in a database.

Object detection was required because OCR alone cannot determine which text belongs to a sensor or cable symbol. The detector therefore provides both the symbol class and the crop coordinates used by the OCR stage. YOLOv5m [25, 12] was selected because it was available in Azure Machine Learning Studio and had been used in related engineering-drawing work.

This chapter focuses on the object detection component of the pipeline. It covers data collection and labelling (Section 4.1), tiling (Section 4.2), post-processing steps (Section 4.4), evaluation metrics (Section 4.3), a data sufficiency analysis (Section 4.5), and the baseline model (Section 4.6). The OCR component is described separately in Chapter 6.

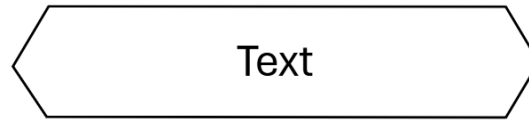


Figure 4.2: A rendition of what a Cable label looks like in an engineering drawing. The "Text" in the middle of the label represents an arbitrary combination of numbers, uppercase letters, and special symbols.

4.1 Data Collection and Labelling

The dataset was collected manually by browsing internal databases and selecting PDFs containing electrical engineering drawings. The labels of interest were cable and sensor symbols; examples of their typical appearance are shown in Figures 4.2 and 4.3.

Data labelling was performed using the Data Labelling tool in Azure Machine Learning Studio. Since the platform does not support native PDF annotation, the engineering drawings were converted to images prior to labelling. In total, 108 PDF documents were converted into 259 page images at 300 DPI, which was sufficient to preserve symbol details in testing. From these, 26 images were selected for local testing to establish a visually diverse subset. Table 4.2 shows the distribution of labels in the test data. The remaining training images (233 in total) were stored in Azure Blob Storage [16] to enable access from the labelling tool.

All images were annotated manually using the bounding-box tool, and all annotations were produced by a single annotator. During labelling, two recurring sources of ambiguity were identified: sensor symbols sometimes resembled table cell borders, and sensor and cable symbols shared similar geometric patterns. These observations influenced two later design choices. First, annotations were collected iteratively to expose the detector to more varied examples of both classes (Section 4.5). Second, class-specific confidence thresholds and Non-Maximum Suppression were used during post-processing to suppress low-confidence and overlapping detections (Section 4.4).

Of the 233 training images, 70 were excluded from labelling because they contained no relevant objects (typically pages with metadata tables or title blocks). The final labelled dataset, therefore, comprised 163 images and 3094 annotations. Table 4.1 summarises the annotation distribution, showing class imbalance with sensors outnumbering cables by approximately 3.3:1.

Finally, annotations were exported in the Azure Machine Learning Tables (MLTable) format, a YAML-based specification required by the YOLOv5 training pipeline in Azure Machine Learning Studio.

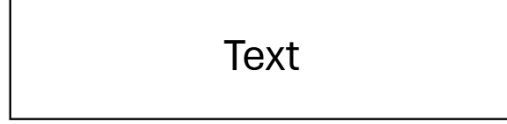


Figure 4.3: A rendition of what a Sensor label looks like in an engineering drawing. The "Text" in the middle of the label represents an arbitrary combination of numbers, uppercase letters, and special symbols.

Label class	Count	Percentage
Sensor	2497	80.70 %
Cable	597	19.30 %

Table 4.1: Dataset composition of the final dataset used for training and validation, and annotation distribution showing substantial class imbalance between sensors and cables.

Label class	Count	Percentage
Sensor	167	66.01 %
Cable	86	33.99 %

Table 4.2: Dataset composition of the held-out dataset used for testing.

4.2 Tiling for Small Object Detection

The engineering drawings often exceed 4000×4000 pixels and, in some cases, reach 9000×9000 , whereas standard object detection models expect much smaller, fixed input resolutions. As a result, training on full-page images requires downsampling, which can remove fine-grained visual features needed to detect small sensor and cable symbols.

The target labels are extremely small relative to the full page. On average, an engineering drawing page covers about 0.49m^2 (approximately 765in^2), while a sensor or cable symbol covers about $1.5 \times 10^{-4}\text{m}^2$ (0.23in^2), or roughly 0.03% of the page. Directly resizing a full high-resolution page to the YOLOv5m input size of 640×640 therefore removes much of the detail needed to recognise these symbols. To preserve local resolution while remaining compatible with the model input size, each page was split into multiple 640×640 tiles and processed tile by tile.

Preliminary experiments without tiling produced very poor detection performance, indicating that full-page downsampling was insufficient for this task. We therefore follow a strategy commonly employed for small object detection, where high-resolution images are split into smaller crops with 25% overlap to preserve effective resolution within each tile [28]. Specifically, each page is divided into an $n \times m$ grid

with 25% overlap between adjacent tiles. This overlap guarantees that objects near tile boundaries are fully contained within at least one tile, reducing detection failures caused by truncation. The main trade-off of this approach is increased computational cost, as both training and inference require processing multiple tiles per original page; however, in this work, it was necessary to obtain reliable detections.

4.3 Evaluation Metrics

For model evaluation, the primary metric in Azure Machine Learning Studio was mean Average Precision (mAP). In the YOLOv5 object detection setup, mAP@0.5 was selected as the primary metric to be maximised. In this context, Average Precision (AP) measures the area under the precision-recall curve for a given class, and mAP is the mean of these AP values over all classes. The "@0.5" notation specifies that detections are counted as correct when their Intersection over Union (IoU) with a ground-truth box is at least 0.5, following the Pascal VOC convention. This corresponds to a moderately strict overlap requirement that penalises clearly misplaced boxes while still accepting minor localisation errors. The metric is reported as mAP@0.5 to distinguish it from COCO-style mAP, which averages AP over multiple IoU thresholds (typically 0.5 to 0.95 in steps of 0.05) and is more commonly used for large, multi-class benchmarks. Given that this work focuses on two symbol classes and on maximising reliable detections for downstream OCR rather than benchmarking across many IoU thresholds, mAP@0.5 provides a simple and interpretable primary measure of detection quality. Precision and recall are defined in Eqs. 4.1 and 4.2.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \quad (4.1)$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \quad (4.2)$$

Here, TP denotes true positives, FP false positives, and FN false negatives. The analysis focuses on class-specific precision and recall, because aggregate metrics can hide class-dependent failure modes (Section 4.5).

4.4 Post-Processing

Raw model predictions are post-processed using a fixed two-step pipeline. First, class-specific confidence thresholds are applied to remove low-confidence detections: 0.5 for cables and 0.7 for sensors. These thresholds show the different precision–recall trade-offs observed for the two classes during validation and are kept fixed in all subsequent

experiments. Second, class-agnostic Non-Maximum Suppression (NMS) with an IoU threshold of 0.2 is applied to eliminate duplicate and overlapping detections, retaining only the highest-confidence prediction per box. This step is essential when the model assigns both class labels (sensor and cable) to the same physical region.

Azure Machine Learning Studio can also apply built-in inference-time filtering (via `box_score_threshold` and `nms_iou_threshold`) before computing validation metrics, so these settings directly affect the reported performance values. These settings were explored during development and are specified where relevant (e.g., in the hyperparameter sweep configuration in Chapter 5). Unless stated otherwise, “post-processing” in this thesis refers to the custom pipeline applied on top of the model outputs.

4.5 Data Sufficiency Analysis

This section documents the iterative development of the detection model and examines how increasing the number of annotated instances was associated with changes in performance. The analysis should be interpreted as an exploratory development study rather than as a controlled ablation experiment, because in addition to dataset size, inference-time filtering and other development choices changed across iterations. Consequently, the observed improvements cannot be attributed solely to the number of annotations, although they do indicate that additional annotated data improved class-wise stability under the developing pipeline. Unless stated otherwise, the custom post-processing pipeline (Section 4.4) is applied only when evaluating models on the held-out test set.

An initial dataset of 1056 annotated instances was collected and used to train a preliminary baseline model and to run a small set of hyperparameter sweeps. Across all dataset versions, the class distribution remains imbalanced, with sensors accounting for roughly three-quarters of the annotations (Table 4.3). With 1056 annotations, the model achieved a validation mAP@0.5 of 0.76, but the class-wise metrics revealed clear asymmetries: cable detections reached moderate precision and recall, whereas sensor detections showed very low precision (0.13) but perfect recall (1.00) (Table 4.4). This demonstrates how aggregate detection metrics can mask class-specific problems, especially when classes differ in visual difficulty or when the dataset lacks sufficient variation.

To reduce these errors, additional data were annotated in subsequent rounds. With 1222 instances, both classes improved substantially (validation mAP@0.5 = 0.96), and sensor performance became more balanced (precision 0.52, recall 0.97), while cable precision and recall also increased. A further increase to 2136 instances led to

more stable, symmetric class-wise performance (Cable: 0.81/0.96; Sensor: 0.87/0.99), with a validation mAP@0.5 of 0.95. Although the mAP@0.5 changed only slightly between 1222 and 2136 annotations, the per-class precision–recall values became more consistent, suggesting that the extra data mainly improved stability and reduced class-specific errors rather than raising the peak score. Based on these observations, data collection was extended once more, and the final labelled dataset used in subsequent experiments comprised 3094 annotated instances (Section 4.1).

Annotations	Class	Count	Percentage
1056	Cable	247	23.39 %
	Sensor	809	76.61 %
1222	Cable	303	24.80 %
	Sensor	919	75.20 %
2136	Cable	464	21.72 %
	Sensor	1672	78.28 %

Table 4.3: Class distribution (counts and percentages) for the datasets used in the data sufficiency analysis.

Annotations	mAP@0.5	Class	Precision (val)	Recall (val)
1056	0.76	Cable	0.49	0.81
		Sensor	0.13	1.00
1222	0.96	Cable	0.91	0.97
		Sensor	0.52	0.97
2136	0.95	Cable	0.81	0.96
		Sensor	0.87	0.99

Table 4.4: Data sufficiency analysis results on validation data.

After applying the custom post-processing steps described in Section 4.4, the models were also evaluated on a held-out test set (Table 4.5). Because the post-processing parameters were fixed using the final pipeline configuration, the test-set comparison across dataset sizes should be interpreted as indicative rather than as a per-model optimal comparison. The test results mirror the main validation trends: the model trained on 1056 annotations suffers from notably low cable recall (many false negatives), the 1222-annotation model substantially reduces both false negatives

and false positives, and the 2136-annotation model further reduces false positives while preserving high recall.

Annotations	Class	Precision (test)	Recall (test)	TP	FP	FN
1056	Cable	1.00	0.14	12	0	74
	Sensor	0.70	0.98	185	79	2
1222	Cable	0.83	0.99	85	18	1
	Sensor	0.88	0.98	183	24	4
2136	Cable	0.99	0.99	85	1	1
	Sensor	0.97	0.98	183	6	4

Table 4.5: Data sufficiency analysis results on test data. Testing is done after the post-processing steps explained in Section 4.4.

4.6 Baseline Model

A baseline object detection model was trained with the YOLOv5m architecture in Azure Machine Learning Studio. All hyperparameters were kept at their default values, except for enabling tiling with a 3×3 grid to better preserve small-symbol detail (Section 4.2). The full labelled dataset was used, with 130 images for training and 33 images for validation. The training data comprised 80%, while the validation data comprised the remaining 20%. The quantitative performance of this baseline detector on the validation and test sets is presented in Chapter 7.1.2.

5. Hyperparameter Optimisation

This chapter explains how the baseline detector was improved through hyperparameter optimisation, aiming to reduce false positives while maintaining high recall. Hyperparameters were tuned using Azure Machine Learning’s random search with early termination to efficiently identify candidate configurations under limited compute.

5.1 Motivation for Hyperparameter Tuning

The baseline model trained on the full dataset (Section 4.6) achieved a high mAP@0.5 but comparatively low precision, with recall close to 1.0 for both classes (see Chapter 7 for detailed metrics). This implies a substantial number of false positive detections and motivates further refinement of the precision-recall trade-off via hyperparameter tuning. Since the downstream OCR step is applied to each predicted bounding box, reducing false positives is important for both computational optimisation and avoiding incorrect text extraction. Hyperparameter tuning was therefore used to improve the precision–recall trade-off without changing the model architecture.

5.2 Hyperparameter Search Strategy

Random search was chosen because it can cover the hyperparameter space efficiently when only a subset of parameters strongly influences performance [5]. In addition, random search integrates naturally with early termination, allowing unpromising trials to be stopped early and resources to be concentrated on better configurations.

To manage computational cost, the Bandit early termination policy was used [18]. Trials were evaluated every 5 epochs, and early stopping was enabled after 10 epochs to prevent termination during the initial unstable training phase. A slack factor of 0.2 was used, meaning that trials performing substantially worse than the current best at the same evaluation epoch were terminated.

5.3 Search Space

The search space was designed around the baseline configuration and focused on parameters expected to affect the precision–recall trade-off. Learning rate and momentum influence how the model learns during training, while the inference-time parameters `box_score_threshold` and `nms_iou_threshold` control which detections are kept before metrics are computed (`box_score_threshold` filters low-confidence detections; `nms_iou_threshold` controls non-maximum suppression). Several parameters (model family and size, image size, batch size, number of epochs, and tiling settings) were kept fixed to establish comparable trials. Based on preliminary experiments, the tiling grid was fixed to 4×3 for all sweep runs (the baseline uses 3×3), to preserve even more detail at the cost of computing overhead. In addition, time constraints and the earlier data sufficiency experiments suggested that some configurations were better kept constant than further tested. Table 5.1 summarises the tuned and fixed hyperparameters used in the sweep.

Hyperparameter	Distribution	Range / Choices	Status
Learning rate	Uniform	[0.01, 0.02]	Tuned
Momentum	Uniform	[0.92, 0.94]	Tuned
Box score threshold	Choice	0.60, 0.62, 0.65, 0.67, 0.70	Tuned
NMS IoU threshold	Choice	0.45, 0.50, 0.55	Tuned
Model name	–	YOLOv5m	Fixed
Model size	–	medium	Fixed
Image size	–	640	Fixed
Epochs	–	80	Fixed
Batch size	–	8	Fixed
Tiling grid	–	4×3	Fixed
Tile overlap	–	0.25	Fixed

Table 5.1: Hyperparameter search space used in the random sweep.

The sweep was configured to run up to 100 trials but constrained by a time budget of 1200 minutes, causing it to terminate when the timeout was reached rather than after all 100 trials had completed; in total, 6 trials finished successfully. Preliminary experiments from the data sufficiency analysis informed the choice of hyperparameters, and comparisons between the tuned baseline model and the default settings indicated that the results from these 6 trials were adequate for selecting candidate models. The

training data comprised 80% of the dataset, with the remaining 20% used for validation.

Although only six trials completed, limiting how thoroughly the search space was explored, most of the models still achieved strong validation performance, suggesting that the random search discovered promising regions given this constraint. The resulting candidate models and their evaluation are presented in Section 7.1.3.

6. Optical Character Recognition

This chapter addresses the problem of extracting sensor and cable identifiers from detected regions using OCR and related methods. The goal is to evaluate different OCR engines and configurations for this task.

Optical character recognition is the task of converting images containing text into machine-readable character sequences. A typical OCR pipeline includes image acquisition, preprocessing (e.g., binarisation, deskewing, and noise removal), layout analysis, character or word recognition, and optional post-processing, such as spell-checking or lexicon-based correction. Classical OCR engines such as Tesseract follow this pipeline and are usually trained on printed or handwritten documents with well-structured layouts. Modern multimodal large language models (MLLMs), also known as vision-language models (VLMs), jointly process pixels and text and can perform OCR as part of a deeper visual understanding task. In this chapter, the focus is on the narrow sub-problem of recognising short alphanumeric IDs in small, automatically cropped regions extracted from engineering drawings, which differs from standard document OCR in both layout and error tolerance requirements.

6.1 OCR Problem Definition and Text Fields

The second part of this thesis is dedicated to extracting information from the bounding boxes predicted by the object detection model. This information consists of sensor and cable IDs rendered as text. The original set of PDF files and exported images does not contain embedded, selectable text; thus, the IDs must be recovered solely from the raster data via OCR. Each ID is located in the centre of a label and consists of a mixture of uppercase letters, digits, and a small set of special characters such as `.`, `-`, or `/`. The IDs do not follow a simple, easily separable pattern within each class and can be very similar in structure and content, making post-processing and error detection more difficult. As a result, even a single-character error can change the meaning of an ID, and simple dictionary-based post-processing is not straightforward.

All IDs are standard Unicode text encoded as ASCII-compatible characters, so the task is purely one of character classification rather than script or language identi-

fication.

6.2 Cropping and Detector-Based Preprocessing

As discussed in Chapter 4, a bounding-box-based detector was used to return coordinates for each predicted sensor and cable. These coordinates are essential for the OCR stage because they define which parts of the image should be cropped and passed to the OCR engine. For each original diagram, the image is therefore cropped into smaller patches that follow the predicted bounding boxes, resulting in views similar to Figures 4.2 and 4.3.

In addition to bounding-box-level cropping, some OCR engines either require or benefit from image preprocessing, such as grayscale conversion, thresholding, or resizing. For each engine, the effect of preprocessing the cropped images on OCR accuracy was evaluated.

6.3 OCR engines

This study focuses on off-the-shelf OCR solutions rather than training a custom recogniser. Training a dedicated OCR model for short, domain-specific identifiers would require a substantially larger and more diverse labelled text dataset than was available in this project, as well as considerable effort in model selection, training, and maintenance. Moreover, the research questions emphasise how different existing OCR engines behave on detector-generated crops and how their errors affect end-to-end identifier extraction, rather than proposing a new recognition architecture.

Traditional open-source OCR engines such as Tesseract [26] remain widely used in both academic and industrial settings. PyTesseract, a Python wrapper for the Tesseract OCR engine, was used; the engine has been open source since 2006 and has historically achieved state-of-the-art accuracy on printed documents. Tesseract performs several preprocessing and layout analysis steps internally, including binarisation and page segmentation, but its defaults are tuned for document-style inputs and can be suboptimal for small, isolated labels. Different page segmentation modes (PSMs) were therefore examined, and the effect of additional external preprocessing on task-specific accuracy was evaluated.

As a commercial baseline, Amazon Rekognition’s `DetectText` API [1] was evaluated. This service can detect and recognise text in images and returns both bounding boxes and recognised strings at the word or line level, without requiring the user to train a model.

Finally, a multimodal GPT-5.0 vision-language model [24] (also referred to as a

multimodal large language model, or MLLM) deployed via Azure AI Foundry [19] was examined. Recent vision-enabled GPT variants can accept images or PDFs as input and produce free-form textual responses, including reading text from complicated scenes or documents. Unlike specialised OCR engines, these models perform OCR as part of a more general reasoning process, which may help in ambiguous cases, yet also introduces less predictable behaviour and higher computational cost.

For each of these models, each predicted bounding box was converted from normalised to pixel coordinates, after which the source diagram was cropped using those coordinates. The cropped patch was then either passed as is or with preprocessing applied. The preprocessing was kept the same across all engines to provide comparable results, including grayscale conversion, resizing, and binarisation.

6.3.1 PyTesseract Configuration

The cropped RGB patches were passed to Tesseract with different page segmentation modes (PSMs), both with and without preprocessing. The default configuration (PSM 3, fully automatic page segmentation) assumes a full page of text, which is poorly matched to small, isolated labels. PSM 7 (single text line) and PSM 9 (single word in a circle) were therefore also evaluated, and the best-performing mode was selected for comparison with other engines.

6.3.2 AWS Rekognition Configuration

For Amazon Rekognition, the `DetectText` API [3] was used via the official `boto3` client library [2]. The cropped patches, both with and without preprocessing, were encoded as PNG in memory and sent as byte arrays. The service returned a list of detected text elements with type (word or line), confidence, and bounding boxes. A wrapper supporting three response aggregation modes was implemented: `LINE` (concatenation of all line-level detections), `WORD` (use of the first detected word), and `ALL` (concatenation of all detected strings). In the main experiments, the `LINE` mode was used.

6.3.3 GPT-5.0 Configuration

For the GPT-5.0 (model version 2025-08-07) experiments, an Azure-hosted multimodal GPT model was accessed programmatically through the Azure OpenAI API using key-based authentication [20]. Each cropped patch, both without and with preprocessing, was encoded as a PNG image, base64-encoded, and included in the request together with the textual prompt: "Extract all text from this image exactly as it appears. Return only the text, nothing else." The model returned a

short textual response, which was treated as the OCR output after leading and trailing whitespace had been removed. Model parameters were kept at their default values.

6.4 Shared Evaluation Protocol and Metrics

All OCR engines were evaluated within a common framework to enforce comparable results. For each test image, detector predictions and a corresponding ground-truth JSON file containing the true bounding boxes and ID strings were loaded. Predictions were matched to ground-truth objects of the same class using IoU with a threshold of 0.5, such that each prediction was paired with at most one ground-truth label.

For every matched prediction, the source diagram was cropped at the predicted bounding box, and the resulting crop was passed to the selected OCR engine. The engine produced a text string, which was compared to the ground-truth ID. It was then recorded whether the OCR output exactly matched the ground truth, together with the predicted class, confidence, IoU, and raw strings. This procedure produced a set of per-prediction results, which were summarised as overall and per-class accuracies. These are calculated using Equation 6.1. In addition, error patterns were analysed by counting empty outputs, extra or missing characters, case-only mismatches, and common single-character confusions. Edit distances were also computed, and failed predictions were manually inspected to identify systematic character confusions.

Two accuracy definitions are reported separately in this thesis. OCR-only accuracy is computed over matched detections,

$$A_{\text{OCR}} = \frac{M}{N_{\text{matched}}}, \quad (6.1)$$

where M is the number of matched detections whose recognised identifier exactly equals the ground truth and N_{matched} is the number of matched detections. End-to-end extraction accuracy is computed over all ground-truth objects,

$$A_{\text{E2E}} = \frac{M}{N_{\text{gt}}}, \quad (6.2)$$

where missed detections count as errors. This distinction is used consistently in all subsequent result tables and discussion.

7. Results

This chapter presents and analyses the results of the object detection and OCR experiments performed on the electrical engineering drawing dataset. The following sections report performance on validation and held-out test sets, describing the experimental setup, metrics, and evaluation outcomes for both the object detection and OCR components.

7.1 Object Detection Results

This section presents the results of the object detection experiments, detailing the experimental setup, baseline, and optimised model performance, and evaluation metrics. Each subsection provides an in-depth analysis of the methods used, summarises the main findings, and discusses their consequences for subsequent OCR processing within the general pipeline.

7.1.1 Experimental Setup

All experiments were conducted on the labelled engineering drawing dataset described in Chapter 4. The full set of 163 annotated images was split into 130 training images and 33 validation images for model development, with a held-out test set of 26 images (273 annotations) reserved for final evaluation. The same test set was used consistently for both object detection and OCR experiments to allow end-to-end analysis.

Training and evaluation were conducted on Azure Compute. The baseline model setup described in Section 4.6 was run on a `Standard_NC4as_T4_v3` instance with 4 vCPUs, 28 GB memory, and one NVIDIA T4 GPU with 16 GB accelerator memory, whereas the hyperparameter sweep described in Chapter 5 was conducted on `Standard_NC8as_T4_v3` instances with 8 vCPUs, 56 GB memory, and one NVIDIA T4 GPU with 16 GB accelerator memory. Model performance was evaluated on the validation set using mAP@0.5 together with class-wise precision and recall as the primary detection metrics, and the post-processing described in Section 4.4 was applied when reporting results on the held-out test set.

7.1.2 Baseline Model Performance

Table 7.1 summarises the baseline performance on the validation set. Using the full dataset, the model achieved a high mAP@0.5 of 0.96, suggesting that the detector generally localised the target symbols under the chosen IoU criterion. This is consistent with the data sufficiency analysis (Section 4.5), where increasing the number of annotated instances improved stability and class-wise performance.

Even with the high mAP@0.5, the baseline precision was comparatively low (0.41 for Cable and 0.32 for Sensor) while recall was close to 1.0 for both classes. Since the downstream OCR step is applied to each predicted bounding box, reducing false positives is important to avoid unnecessary OCR processing and to limit incorrect text extraction.

The per-class precision-recall curves on the validation set in Figures 7.1 and 7.2 provide additional understanding into this behaviour by comparing the baseline and tuned detectors. For the baseline model, both classes operate in a high-recall but lower-precision regime: for the cable class, precision remains high over a large part of the recall range, but drops sharply as recall approaches 1, indicating that the model produces more false positive detections when attempting to recover nearly all true instances. For the sensor class, the same general pattern is visible, although the decline in precision is more gradual before becoming pronounced at the highest recall levels. The tuned model shifts these curves upward, especially in the high-recall region, achieving higher precision at comparable recall levels for both classes. These curves therefore support the quantitative results by showing that the baseline model is not primarily limited by missed detections, but rather by reduced precision under very high recall, and that hyperparameter tuning mitigates this by reducing false positives without sacrificing recall.

mAP@0.5	Class	Precision	Recall
0.96	Cable	0.41	0.98
	Sensor	0.32	0.995

Table 7.1: Baseline model performance on the validation set.

In addition to validation performance, the baseline model was evaluated on the held-out test set after applying the post-processing pipeline described in Section 4.4 using the same class-specific confidence thresholds as the final model. Table 7.2 reports class-wise precision, recall, and counts (TP/FP/FN) under this post-processed configuration. As expected, post-processing substantially reduces false positives compared to the default-threshold validation results in Table 7.1, resulting in higher precision

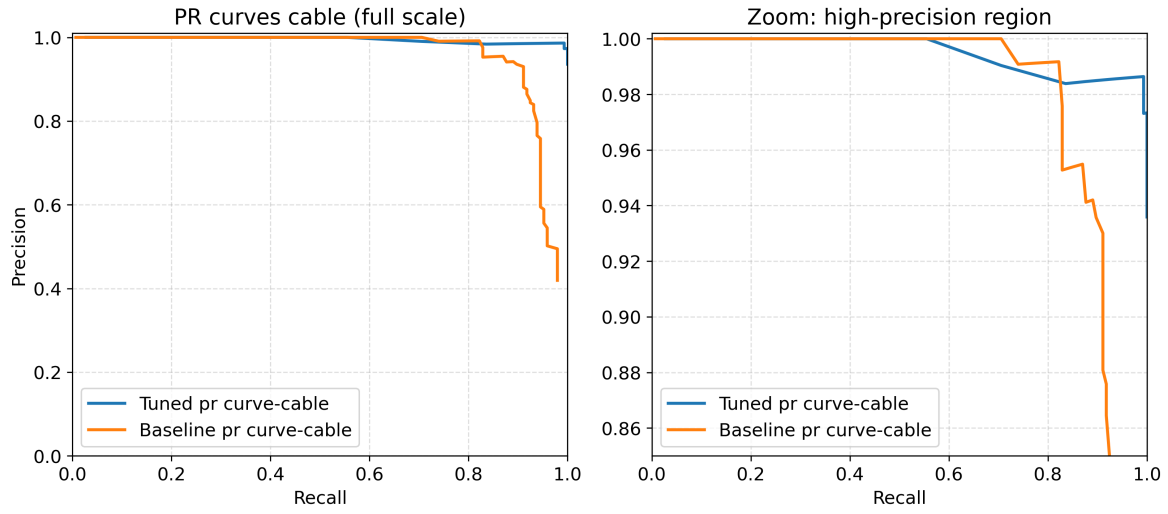


Figure 7.1: Precision-recall curves for the cable class for the tuned and baseline models. The full curves are shown on the left, and a zoomed view of the high-precision region is shown on the right. The tuned model achieves a more favorable precision-recall trade-off, especially at high recall, where the baseline model exhibits a marked decline in precision.

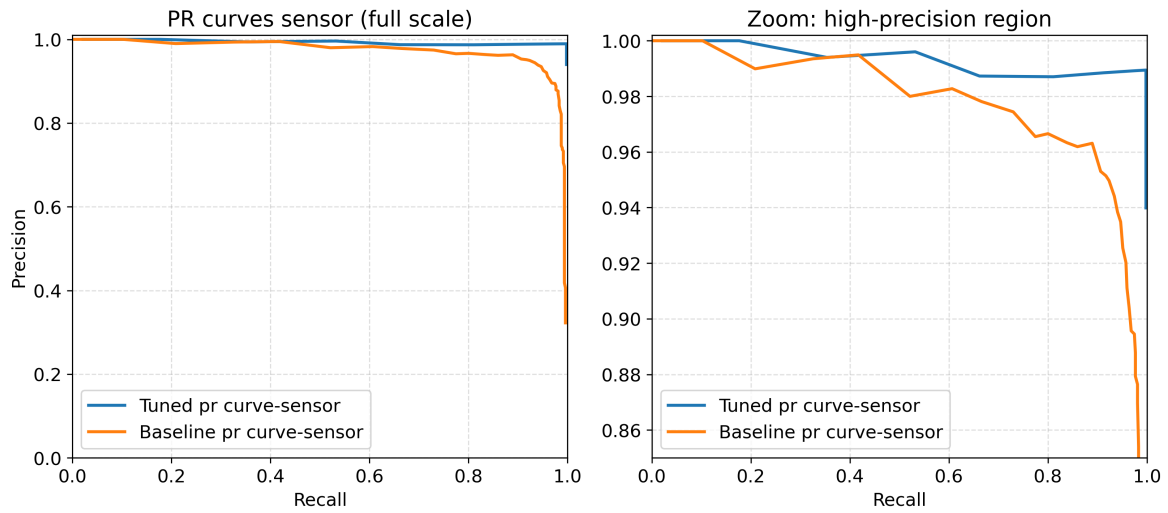


Figure 7.2: Precision-recall curves for the sensor class for the tuned and baseline models. The full curves are shown on the left, and a zoomed view of the high-precision region is shown on the right. The tuned model provides a more stable precision-recall trade-off, notably at high recall, where the baseline model exhibits a more pronounced decrease in precision.

while retaining high recall.

Class	Precision	Recall	TP	FP	FN
Cable	0.97	0.98	84	3	2
Sensor	0.95	0.99	186	9	1

Table 7.2: Baseline model performance on the held-out test set, including post-processing.

7.1.3 Hyperparameter Optimisation Results

Hyperparameter optimisation produced two competitive configurations (Model A and Model B). The two finalist models shared most settings and differed only in the score threshold, learning rate, momentum, and NMS IoU threshold, as summarised in Table 7.3.

Hyperparameter	Model A	Model B
Box score threshold	0.67	0.62
Learning rate	0.01639	0.01525
Momentum	0.9231	0.9337
NMS IoU threshold	0.50	0.55

Table 7.3: Hyperparameters that differed between the two finalist models after optimisation. All other settings were identical: image size 640, model YOLOv5-medium, 80 epochs, tile grid size 4×3 , tile overlap ratio 0.25, batch size 8, and validation IoU threshold 0.50.

Validation metrics were used to shortlist two candidate configurations (Model A and Model B), as both achieved strong detection performance and comparable class-wise behaviour. The final selection criterion prioritised recall, and especially the avoidance of false negatives, because missed detections cannot be recovered by downstream OCR. Under this criterion, Model B was the more suitable recall-oriented candidate on validation: it achieved perfect recall for the cable class (1.00 vs. 0.99 for Model A) while matching Model A on sensor recall (0.99 for both models), although Model A achieved slightly higher validation mAP@0.5 and precision.

The shortlisted models were also compared on the held-out test set under the fixed custom post-processing pipeline (Section 4.4) to assess their practical behaviour in the end-to-end setting. In this comparison, Model B produced no false negatives for either class, whereas Model A produced one false negative for the sensor class.

The improved test-set behaviour of Model B is plausibly explained by its slightly less restrictive detection settings. In particular, Model B used a lower box score thresh-

Metric	Model A	Model B
<i>Validation metrics (proportion)</i>		
Val mAP@0.5	0.97	0.96
Val Precision (C)	0.96	0.94
Val Recall (C)	0.99	1.00
Val Precision (S)	0.95	0.94
Val Recall (S)	0.99	0.99
<i>Test set errors (count)</i>		
Test FP (C)	0	1
Test FN (C)	0	0
Test FP (S)	3	1
Test FN (S)	1	0
Test FP total	3	2
Test FN total	1	0

Table 7.4: Comparison of the two finalist models. Validation metrics were used to shortlist the candidates under a recall-oriented selection criterion, and held-out test-set results are included to illustrate their practical behaviour under the fixed post-processing pipeline (26 images, 273 annotations). Validation values are reported as proportions, and test results are reported as counts. Abbreviations: Val = validation, FP = false positives, FN = false negatives, C = Cable, S = Sensor.

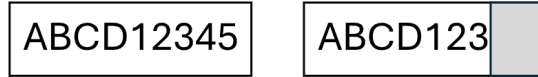


Figure 7.3: Illustration of right-edge truncation in a detector crop. Left: intended crop covering the full identifier. Right: overly tight crop in which the rightmost characters are excluded (grey region), which can cause downstream OCR to return an incomplete identifier.

old (0.62 vs 0.67), which likely increased recall by retaining lower-confidence detections that Model A may have discarded. Model B also used a slightly higher NMS IoU threshold (0.55 vs 0.50), making suppression of overlapping boxes less aggressive and potentially preserving borderline detections.

A minor qualitative limitation found across trained models, including the final configuration, was rare truncation of predicted boxes at the right image border (see Figure 7.3). This can lead to partial string capture in the downstream OCR step and was present in the raw model outputs (prior to NMS), suggesting it originates from the detector rather than the post-processing pipeline. On the test set, there were 2 instances of cropping being too aggressive, thus a very small percentage. Mitigation strategies for this issue are discussed in more detail in Section 7.3. Nevertheless, model B was chosen as the final pipeline candidate, and no further model training was deemed

necessary within the scope of this thesis. Beyond scalar metrics, it is also useful to examine how tuning changed the full precision–recall behaviour.

Figures 7.1 and 7.2 show the effect of hyperparameter tuning on the precision–recall trade-off by plotting the validation precision–recall curves of the baseline detector and the final tuned model (Model B) for cable and sensor symbols, respectively. In both classes, the tuned model closely tracks the high-recall region of the baseline while shifting the precision–recall trade-off towards higher precision, especially in the high-recall range that is most relevant for downstream OCR. These curves therefore show that hyperparameter tuning improved the precision–recall trade-off relative to the baseline and that the selected tuned model retains the high-recall behaviour required for the downstream OCR stage. Model B was therefore retained as the final detector for the OCR pipeline, as it best matched the recall-oriented selection criterion used in this thesis.

7.2 OCR Results

For all OCR experiments, the tables report a subset of representative configurations, and the text discusses the full set of tested settings and their behaviour. In every case, OCR is applied to crops generated from the final detector’s predictions rather than from ground-truth boxes, so the reported accuracies represent realistic downstream performance in the studied setting.

7.2.1 PyTesseract Results

PyTesseract performed poorly in this setting (Table 7.5). Without preprocessing or configuration, Tesseract frequently confused visually similar characters: 1 with I (13 occurrences), T with E (6 occurrences), and 0 with O (5 occurrences). This behaviour is expected because Tesseract’s default page segmentation mode (PSM 3) assumes a full page of text rather than isolated labels. Here, accuracy is the ID-level exact-match rate defined in Equation 6.1, computed as the fraction of matched predictions in which the OCR output string exactly equals the ground-truth ID. In addition, there were two cases where OCR failed to extract text correctly due to box cropping (See Section 7.1.3 and Figure 7.3).

Specifying the page segmentation mode (PSM) improved the accuracy considerably. PSM 7 (single text line) and PSM 9 (single word in a circle) both outperformed the default, with PSM 9 achieving the best results for sensors at 51.9% accuracy. This parameter is important because explicitly specifying the layout helps the engine focus on the relevant content.

Class	PSM 3	PSM 7	PSM 9
Sensor	2.1%	39.0%	51.9%
Cable	8.1%	16.3%	11.6%

Table 7.5: OCR accuracy (Equation 6.1) using PyTesseract with different page segmentation modes. Default uses PSM 3 (fully automatic page segmentation). PSM 7 treats the image as a single text line. PSM 9 treats the image as a single word in a circle. Here, the accuracy is reported for the plain patches only, as the pre-processed patches performed even worse than PSM 3.

Traditional preprocessing techniques such as grayscale conversion, thresholding, and resizing did not improve Tesseract’s recognition accuracy. In fact, although the preprocessed images appeared visually clearer to human observers, using these methods reduced OCR performance. This finding shows that external preprocessing can interfere with Tesseract’s built-in image processing pipeline, leading to less desirable or conflicting transformations. Therefore, external preprocessing can unintentionally degrade recognition accuracy because Tesseract has already performed internal operations.

7.2.2 AWS Rekognition Results

AWS Rekognition performed better than PyTesseract. The results of sending plain patches without preprocessing to the OCR engine are shown in Table 7.6. Rekognition achieved high ID-level accuracy without any task-specific tuning or preprocessing, suggesting that the underlying model is well-suited for short alphanumeric strings. Pre-processing the patches reduced the accuracy to nearly 0%.

Class	Accuracy
Sensor	96.8%
Cable	97.7%

Table 7.6: OCR accuracy (Equation 6.1) using AWS Rekognition without any preprocessing.

Additional investigation into where AWS Rekognition failed revealed two cases where OCR failed to extract text correctly due to box cropping (See Section 7.1.3 and Figure 7.3). In two additional cases, the bounding boxes were correct, but Rekognition misclassified characters in IDs with two values separated by /, predicting I instead of / and L instead of 4. These errors are consistent with common OCR errors caused by visually similar or difficult characters.

7.2.3 GPT-5.0 Results

Table 7.7 summarises the class-wise ID-level accuracies of GPT-5.0 with preprocessing applied. Interestingly, unlike PyTesseract and Rekognition, applying the preprocessing steps described in Section 6.3, modestly improved overall accuracy from 80.2% to 82.4%.

Class	Accuracy
Sensor	84.0%
Cable	75.6%

Table 7.7: OCR accuracy (Equation 6.1) using GPT-5.0, with pre-processing steps taken to improve the quality of the extraction.

Similar character confusions to those observed with PyTesseract (Subsection 7.2.1) also occurred with GPT-5.0, but at a lower frequency, and two cases where GPT-5.0 failed to extract text correctly due to box cropping (See Section 7.1.3 and Figure 7.3). In terms of runtime, this approach was the slowest of all systems tested and achieved lower ID-level accuracy than AWS Rekognition. Moreover, because it is a general-purpose chat model with stochastic outputs, its predictions are less deterministic and therefore require careful validation before use in a production pipeline.

7.2.4 OCR Engine Comparison

When choosing a primary OCR engine, the priority was placed primarily on accuracy. This decision was taken to achieve the most accurate OCR predictions possible, to prevent introducing false information or, in the worst case, information loss later in the pipeline. For this task, AWS Rekognition was therefore selected as the main engine: it was the fastest and most accurate option and did not require additional preprocessing or parameter tuning to achieve good results.

7.3 Automatic Bounding Box Quality Assessment for OCR Optimisation

In rare cases, the detection model produced bounding boxes that were cropped too aggressively on the right-hand side of the label, truncating the ID (see Figure 7.3 and Section 7.1.3). To reduce information loss, or at least make users aware of incomplete outputs, we investigated two mitigation strategies: (i) Adding padding to the right side

of the bounding box when deemed necessary, (ii) Using an external database of known IDs to cross-reference OCR outputs.

Both approaches have clear limitations. Padding can unintentionally include neighbouring symbols or graphical elements to the right of the label, causing the OCR engine to output extra characters that are not part of the true ID. Database-based correction assumes that the stored IDs are correct, up to date, and relevant to the specific diagram; if this assumption fails, the database may introduce systematic errors rather than correct them.

These strategies were investigated experimentally but are not part of the final deployed pipeline, as their impact on accuracy was negative.

7.3.1 Motivation and Problem Statement

During evaluation of the detection model, a small number of predicted bounding boxes were found to truncate text content at their right-hand boundaries (Figure 7.3). In these cases, the subsequent OCR stage received only a partial ID, which directly reduced end-to-end extraction accuracy. Because errors at this stage propagate to all later steps, even rare truncations are problematic in a pipeline that aims to populate a reusable identifier table.

The experiments in this section investigate whether simple automatic heuristics can detect such truncation and trigger corrective actions before OCR. The main idea is that, if potentially truncated boxes can be identified automatically, small, targeted adjustments to their coordinates might recover missing characters without retraining the detector. In addition, an internal ISO code database was explored as a possible aid for flagging implausible OCR outputs and guiding box extensions. Simple rule-based pattern matching or regular expressions were not applicable here, because the IDs do not follow a single, well-defined format (Section 6.1).

7.3.2 Investigated Approaches: Padding

Several approaches for automatically detecting potentially truncated bounding boxes were analysed, each based on different assumptions regarding the visual and statistical characteristics of truncation.

The first approach assumed that truncated text leaves detectable visual irregularities at image boundaries. When a bounding box cuts through text content, incomplete characters or cut-off strokes should produce high pixel intensity or strong edge responses at the crop boundary. The density of dark pixels and edge responses within narrow margins at the right boundaries of each crop was analysed, and boxes in which these values exceeded some arbitrary thresholds were flagged.

The second approach assumed that truncated bounding boxes would exhibit unusual geometrical characteristics compared to correctly detected boxes of the same class. The mean and standard deviation of predicted bounding-box widths were computed for each class type from the respective test images, and test boxes whose dimensions deviated substantially from these statistics were flagged. Unusually narrow boxes were considered candidates for horizontal truncation and thus for expansion.

The third approach used the OCR service’s confidence scores for each detected text element. It was hypothesised that truncated text would lead to lower confidence scores, specifically for words whose boundaries were such that characters might be incomplete. Boxes were flagged when their confidence scores fell below a threshold.

For all detection approaches, identified truncation candidates were addressed by expanding the bounding box horizontally on the right edge before OCR processing. Padding amounts ranging from 0.01% to 0.02% of the image width were applied to the flagged boundaries to capture any text that might have been sliced off by the original detection.

The experimental results were negative for all approaches. Table 7.8 summarises the impact of applying padding to boxes flagged by the best performing detection method, which was the statistical outlier method.

Table 7.8: Impact of automatic padding on OCR accuracy

Metric	Value
Baseline accuracy (no padding)	97.1 %
Boxes flagged for padding	29
Cases improved by padding	3
Cases degraded by padding	26
Net change	-23
Final accuracy with padding	74.8 %

Although the padding strategy successfully identified and corrected the three true truncation cases, it caused 26 previously correct results to become incorrect, reducing the number of correct extractions by 23 and lowering accuracy from 97.1% to 74.8%. A closer analysis of the cases revealed several systematic problems with the automatic padding approach. The most common issue was the capture of adjacent text content whenever padding was applied. The symbols are sometimes tightly packed, so even small amounts of padding frequently extend the crop region into neighbouring symbols, causing the OCR system to return concatenated content from multiple fields rather than the intended single field. When padding is incorrectly applied to a correctly cropped box (a false positive), the result is almost always degraded because additional

content is introduced. However, when truncation is missed (false negative detection), the original result remains, which may still be partially correct.

All three detection strategies were further limited by the fact that legitimate text content frequently appeared near crop boundaries. Delimiters, punctuation, and boundary characters were often positioned at or near the edges of correctly cropped boxes, so the resulting patterns in pixel intensities, bounding-box dimensions, and OCR confidence were difficult to distinguish from genuine truncation artefacts. In the current use case, the extracted classes and IDs form a reusable table that internal systems use to search for components across drawings and to trace connections, reducing the need for manual page-by-page inspection, which makes reliable identification of truncation particularly important. However, the methods investigated here could not achieve this: statistical outlier detection was ineffective because truncated boxes were not geometric outliers, the loss of one or two characters did not substantially change bounding-box dimensions compared to natural variation in content length, and OCR confidence scores showed only a weak correlation with truncation.

7.3.3 Investigated Approaches: Database

This method compared OCR outputs with an internal ISO code database, with the aim of identifying and correcting wrongly cropped bounding boxes or misinterpreted text. The process began by extracting text from cropped bounding boxes using AWS Rekognition. Each predicted text string was then compared to the ISO codes. If there was an exact match, the detection was classified as valid; if the detection was not in the database, string similarity metrics found the closest ISO code. The pipeline also checked whether the OCR output was a truncated version of a valid code by testing whether it was a prefix of any known ISO code. When truncation was suspected, the average character width was estimated using the bounding-box size and text length, the box was extended to the right by the estimated number of missing characters, and the enlarged area was re-processed with OCR to recover any omitted text.

The ISO code database was programmatically imported from an updated Excel file providing a relatively thorough and up-to-date list of valid codes. Each OCR detection was cross-referenced against this database, and entries not present in it were flagged for further analysis.

This approach did not succeed for two main reasons. First, although the ISO code database had been recently updated, it did not include all the codes present in the engineering drawings, so some valid detections were incorrectly classified as failures. Second, in cases involving wrongly cropped bounding boxes, the resulting OCR strings sometimes matched valid but incorrect codes in the database. These

cases were therefore treated as correct by the validation logic, and no corrective action was taken, introducing a risk of reinforcing systematic misreadings instead of detecting them.

7.3.4 Automatic Bounding Box Quality Assessment Conclusions

Unfortunately, no tested approach reliably improved OCR accuracy; rather, all methods led to a notable decrease in performance. The main difficulties stemmed from variability in bounding box widths and the absence of extensive, up-to-date code databases for effective comparison and validation. Taken together, these observations indicate that the evaluated strategies are not viable for improving OCR extraction reliability in the current context. In practice, this limitation has consequences for the deployment of automated document processing in engineering workflows: engineers cannot assume that extracted IDs are consistently accurate, so manual review will stay essential to determine the integrity of the extracted information. More generally, this persistent risk of cascading errors shows an important constraint on the adoption of end-to-end automated extraction pipelines in technical domains. Until a more consistent bounding box detection model is trained, or more exhaustive code databases are available, downstream processes must explicitly account for the residual uncertainty present in these automated extraction results.

7.4 Final Pipeline Results

This section summarises the end-to-end performance of the final pipeline (See Figure 4.1), combining the selected detection model (Model B) with AWS Rekognition as the primary OCR engine. The goal is to quantify how well the system, as a whole, extracts sensor and cable identifiers from engineering drawings under realistic conditions, where OCR operates on detector-generated crops.

7.4.1 Evaluation Setup

The final pipeline was evaluated on the held-out test set of 26 engineering drawing pages comprising 273 annotated symbols (167 sensors and 86 cables), which was not used during model development or hyperparameter tuning. For each page, Model B produced sensor and cable detections, post-processed using the fixed class-specific thresholds and cross-class non-maximum suppression described in Section 4.4. The resulting bounding boxes were then converted to pixel coordinates and used to crop the original images,

and each crop was passed as-is (without additional preprocessing) to AWS Rekognition’s `DetectText` API.

Detector predictions were matched to ground-truth symbols of the same class using an IoU threshold of 0.5, guaranteeing at most one prediction per ground-truth object, as in the OCR evaluation protocol described in Section 6.4. For each matched pair, the pipeline produced an extracted ID string, which was compared to the ground-truth ID at the exact-match level. IDs with no corresponding detection were treated as missed, and predictions without a matching ground-truth object were recorded as false positives at the detection stage.

7.4.2 Detection Component Summary

On the held-out test set, the final detector (Model B) achieved high precision and recall for both symbol classes after post-processing. As reported in Table 7.4, Model B produced no false negatives for either sensors or cables on the test set, while keeping false positives low (one cable and one sensor false positive across all pages). In terms of class-wise metrics, both precision and recall were at or near 1.00 after applying the tuned thresholds and non-maximum suppression.

This behaviour stems from the design decision to prioritise recall at the detection stage: missed detections cannot be recovered by downstream OCR, whereas a small number of false positives mainly increases OCR workload and can usually be mitigated later. The rare qualitative issue of aggressive cropping at the right-hand boundary affected only two detections in the test set, so its impact on detection-level metrics was minimal.

7.4.3 OCR Component Summary

When applied to detector-generated crops, AWS Rekognition achieved high ID-level accuracy for both symbol classes. As shown in Table 7.6, Rekognition correctly recognised approximately 96.8% of sensor IDs and 97.7% of cable IDs on the test set when using plain, unprocessed patches. Traditional OCR (PyTesseract) and GPT-based OCR were markedly less accurate under the same conditions, and additional preprocessing steps tended to reduce, rather than improve, performance for Rekognition.

Inspection of the remaining OCR errors revealed two main patterns. First, a small number of failures were caused by tight detector crops that truncated the rightmost characters of an ID, leaving Rekognition with incomplete input. Second, some errors arose from confusing visually similar characters, such as 1/I, 0/O, or 4/L, specifically in IDs containing two values separated by /. These patterns are consistent with common OCR failure modes reported in earlier work on technical drawings and equipment tags.

7.4.4 End-to-End ID Extraction Accuracy

Combining detection and OCR, the final pipeline correctly extracted approximately 97–98% of all sensor and cable IDs on the held-out test set. In other words, for nearly all ground-truth symbols, the pipeline produced a bounding box of the correct class and an OCR output string that exactly matched the ground-truth identifier (Equation 6.2). The few remaining errors consisted of: (i) IDs affected by aggressive cropping at the right-hand boundary, leading to missing trailing characters, (ii) IDs misread due to character-level confusions in the OCR stage; and (iii) two false positive detections that produced spurious IDs not present in the ground truth.

From an engineering perspective, this level of performance means that the pipeline can substantially reduce manual effort when locating symbols and cross-referencing IDs across drawings, while still leaving room for human review in high-impact or ambiguous cases. The extracted classes and IDs form a reusable database table that internal systems can query for search, traceability, and label verification tasks. In the deployed setting, the detection-OCR pipeline is embedded in an Extract, Transform, and Load (ETL) process that reads engineering drawings from internal databases, converts them into images, runs the trained model and OCR on the resulting pages, and writes the extracted identifiers into a dedicated database table. This table is then consumed by existing internal tools, so that symbol lookups and cross-diagram checks can be performed through standard query interfaces rather than manual page-by-page inspection.

7.4.5 Implications for Deployment

Although the final pipeline achieves high end-to-end accuracy in the studied setting, the experiments on automatic bounding-box correction suggest that naive post-hoc fixes could easily degrade performance. Heuristic padding and database-driven correction strategies failed to improve, and in some cases substantially reduced the ID-level accuracy by introducing additional errors when applied too broadly. As a result, the deployed pipeline intentionally omits these correction mechanisms and instead relies on the tuned detector and Rekognition’s baseline OCR.

In practice, this means that the system is best used as an assistive tool rather than as a fully autonomous replacement for human judgement. The pipeline can pre-populate identifier tables and indicate likely symbol locations, but engineers should remain aware of residual uncertainty, especially for safety-critical decisions or for drawings that differ substantially from the training distribution. Within these constraints, the final pipeline demonstrates that near-complete automatic extraction of sensor and cable IDs is feasible for internal Wärtsilä drawings using off-the-shelf components and moderate annotation effort.

8. Discussion

This thesis investigated whether sensor and cable identifiers can be automatically detected, read, and stored from raster engineering drawings. The results show that a two-stage detector-OCR pipeline can achieve high accuracy on the studied Wärtsilä drawings and produce structured IDs suitable for downstream search, traceability, and label verification. The practical contribution is an automated extraction workflow for a task that previously required manual inspection of multiple diagrams, while the methodological contribution is an evaluation of how detection quality, OCR choice, and their interaction affect final ID-level accuracy.

For the detection part (RQ1), tiling turned out to be a critical design choice. When training was performed on full pages, performance was poor, consistent with prior observations of small objects in large images. Once the pages were tiled into smaller segments, the model could distinguish the symbols more clearly, and both precision and recall improved. The data sufficiency analysis further showed that adding more annotated symbols increased mAP@0.5 slightly and also corrected class-specific issues, such as very low precision for sensors in the first dataset versions, thus answering RQ2.

Compared with recent YOLOv7-based systems that use multiple detectors to segment views and recognise diverse annotation types [15], this thesis addresses a narrower detection problem with two classes, sensors and cables, and uses a single YOLOv5m detector with explicit tiling of high-resolution pages [28]. The comparison is therefore indirect: broad drawing-understanding pipelines evaluate many annotation types and layout elements, whereas this thesis evaluates whether a constrained set of symbols can be localised accurately enough for exact ID extraction. In that narrower setting, the final detector achieves precision and recall close to 0.99–1.00 on the held-out test set after post-processing, showing that a tightly scoped detector can be highly reliable when the target symbol set and drawing style are constrained.

Hyperparameter tuning produced six viable candidate detectors, of which two were shortlisted based on validation metrics. The final model was selected to minimise false negatives, consistent with the decision to prioritise recall for downstream OCR: missed symbols cannot be recovered later, whereas a small number of false positives

can be filtered or inspected downstream. With the chosen model and post-processing, both sensor and cable symbols achieved high precision and recall on the validation and held-out test sets. This thesis did not aim to exhaustively compare detector families. Instead, it shows how tiling, annotation volume, threshold selection, and NMS affect whether a YOLO-based detector is reliable enough to serve as the first stage of an ID-extraction pipeline.

For the OCR component (RQ3), the experiments revealed clear performance differences between the engines. PyTesseract struggled with the small, densely packed alphanumeric IDs, even after testing several page segmentation modes and simple pre-processing techniques. AWS Rekognition, by contrast, performed well on the cropped labels, achieving approximately 97–98% correct IDs without preprocessing. These results suggest that, in this setting, modern cloud-based OCR is better suited to short technical labels than a classical OCR engine. GPT-based OCR was usable but slower and less accurate than Rekognition, and is therefore not an ideal choice as the primary engine in a production pipeline. Villena Toro and Tarkian’s eDOCr2 framework [27] combines multiple text detectors, OCR engines, and image processing steps to achieve a character error rate below 1% and a character recall of 99.27% on their benchmark. By comparison, the present thesis reaches approximately 97–98% ID-level exact-match accuracy using off-the-shelf OCR on detector crops without a custom OCR architecture or domain-specific recogniser. Although these metrics are not directly comparable, because character-level and ID-level evaluation capture different failure modes and rely on different datasets, the comparison still suggests that strong practical performance can be achieved with a much simpler OCR stage when the task is restricted to short labels in a controlled industrial setting.

The end-to-end experiments show how errors accumulate across pipeline stages. While the pipeline achieves an extraction accuracy of approximately 97–98% for sensor and cable IDs under the tested conditions, similar to or above the rates reported in some recent work on technical drawings [23, 27], this level of performance does not remove the risk of cascading errors. Even rare instances of missed or misread IDs, for example due to truncated bounding boxes or common single-character confusions such as 0/O or 1/I, propagate into the final dataset and can lead to missing or incorrect entries. Attempts to reduce these errors via automated methods, such as boundary padding or database-driven correction, can amplify errors when applied too broadly or with inaccurate assumptions (RQ4). The nuclear engineering IDQ system by Hammad et al. [10] similarly shows that segmentation, colour inversion, and noise filtering can improve tag coverage in plant drawings, but that regular-expression-based filtering and merging of multiple OCR backends must be carefully tuned to avoid false positives.

Khan et al. [13] propose a hybrid vision-language framework in which a rotation-

aware YOLOv11-OBB detector is combined with fine-tuned Donut and Florence-2 models to parse nine types of annotations in 2D mechanical drawings, achieving a reported F1-score of 94% and a hallucination rate of 10.8% on their curated dataset. In contrast, the present thesis adopts a simpler two-stage pipeline with a YOLOv5m detector and a single commercial OCR engine, restricted to two symbol classes. Relative to Khan et al., this narrower focus trades generalisability for tighter control over a specific industrial task, although direct comparison of the metrics remains difficult due to differences in the benchmarking data.

There are several limitations to keep in mind. The dataset is relatively small, sourced from a single company, and limited to two symbol classes with clear class imbalance. The held-out test set is also modest in size, so rare failure modes are unlikely to be fully captured by the reported metrics. In addition, the detector is based on a single model family, YOLOv5m, without comparison to newer architectures or segmentation-based methods that might better capture complex symbol boundaries. On the OCR side, no successful custom post-correction system was implemented, and external engineering databases were only used in limited experiments rather than as an integrated part of the final pipeline.

The reported accuracy should therefore not be interpreted as directly transferable to other companies, drawing standards, symbol conventions, or scanning conditions. However, the model may perform well on drawings that are similar to the training data in terms of symbol design, font style, image quality, and drawing layout. Because engineering drawings are often proprietary, evaluating this transferability on external datasets is difficult; broader generalisation would therefore require access to additional drawings from other projects, vendors, or industrial environments. What is more likely to generalise is the pipeline design pattern: preserving small-symbol detail through tiling, tuning the detector with the downstream OCR task in mind, evaluating OCR on detector-generated rather than ideal crops, and reporting end-to-end exact-match accuracy. In settings with different fonts, noise levels, symbol styles, or safety requirements, the same workflow would require new annotations, validation data, and threshold calibration before deployment.

Despite these limitations, the results show that automatic extraction of sensor and cable IDs from engineering drawings is feasible with current tools. In the tested use case, the pipeline reduces manual work when engineers need to locate components or trace connections across multiple documents and provides a basis for internal search tools built on the extracted identifier table. Given the remaining errors and the context-specific evaluation, however, the system should be deployed as an assistive tool rather than as the sole source of information, especially in safety-critical contexts. Future work should focus on broader validation across drawing standards and vendors, improved

bounding-box refinement, domain-specific OCR post-correction, and active learning strategies for improving annotation coverage and class balance.

9. Conclusions

This thesis set out to investigate whether sensor and cable information can be automatically extracted from engineering drawings using object detection and OCR, and whether the extracted IDs are accurate enough to be stored and reused in downstream engineering workflows. On the studied Wärtsilä drawings, a YOLOv5m-based detector with tiling and tuned post-processing attained high precision and recall for both sensor and cable symbols, and AWS Rekognition correctly recovered most IDs from the detected crops. In the studied use case, the extracted classes and IDs form a reusable table that internal systems use to search for components across drawings and to trace connections, reducing the need for manual page-by-page inspection and forming the basis of an automated ETL process for engineering documentation.

Combined, the object detection and OCR steps resulted in an end-to-end pipeline that correctly extracts around 97–98% of ground-truth sensor and cable IDs on a held-out test set. In practice, this means that the system can substantially reduce manual effort for locating and cross-referencing identifiers, while still leaving room for human checking in high-impact cases. The main contributions of the thesis are: (i) an end-to-end pipeline tailored to this industrial use case, (ii) a data sufficiency analysis for small engineering symbols, (iii) a practical comparison of several OCR engines on short technical IDs, and (iv) an evaluation of how bounding-box truncation errors and simple post-hoc correction strategies affect end-to-end identifier extraction accuracy.

These findings should nevertheless be interpreted within the limits of the studied context: the dataset is small, drawn from a single organisation, and restricted to two symbol classes, while the pipeline relies on one detector family and one OCR service. Accordingly, the reported performance cannot be assumed to transfer directly to other industrial settings without further validation, and occasional cascading errors remain a practical concern. The thesis also shows that automatic bounding-box refinement strategies based on padding and database-driven correction can substantially degrade OCR accuracy in this setting, reinforcing the need for cautious quality control in end-to-end pipelines.

Overall, the work demonstrates that careful design of tiling, data collection, post-processing thresholds, and OCR selection can be as important as the choice of detector

family when building practical extraction systems for small, high-value identifiers in engineering drawings. The present pipeline should be understood as a validated design pattern for one industrial context rather than as a universally transferable model. The same principles could be extended to a wider range of engineering symbols and drawings from other industrial settings, provided that additional data collection, careful evaluation, and domain-specific adaptation are carried out. Future work should focus on improved bounding-box refinement, domain-specific OCR post-correction, and broader evaluation on more diverse drawing sets. With such extensions, similar pipelines could become more robust for engineering documentation workflows, where automated extraction and ETL routines support, but do not replace, human oversight.

Bibliography

- [1] Amazon Web Services. Amazon Rekognition. <https://docs.aws.amazon.com/rekognition/latest/dg/what-is.html>. Accessed: 12-2-2026.
- [2] Amazon Web Services. Boto3. <https://docs.aws.amazon.com/boto3/latest/>, 2026. Accessed: 20 February 2026.
- [3] Amazon Web Services. DetectText. https://docs.aws.amazon.com/rekognition/latest/APIReference/API_DetectText.html, 2026. Accessed: 20 February 2026.
- [4] Y. Bai and X. Xu. Object boundary encoding: a new vectorisation algorithm for engineering drawings. *Computers in industry*, 46(1):65–74, 2001.
- [5] J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *Journal of machine learning research : JMLR.*, 13:281–305, February 2012.
- [6] D. Dori and Y. Velkovitch. Segmentation and recognition of dimensioning text from engineering drawings. *Computer vision and image understanding*, 69(2):196–201, 1998.
- [7] E. Elyan, C. M. Garcia, and C. Jayne. Symbols classification in engineering drawings. In *2018 International joint conference on neural networks (IJCNN)*, pages 1–8. IEEE, 2018.
- [8] E. Elyan, L. Jamieson, and A. Ali-Gombe. Deep learning for symbols detection and classification in engineering drawings. *Neural networks*, 129:91–102, 2020.
- [9] L. Fletcher and R. Kasturi. A robust algorithm for text string separation from mixed text/graphics images. *IEEE transactions on pattern analysis and machine intelligence*, 10(6):910–918, 1988.
- [10] I. Hammad, M. de Costa, A. Boroomand, and M. Anwar. Automating equipment identification in nuclear engineering drawings. *Nuclear Engineering and Design*, 436:114002, 2025.

- [11] R. Helle. Intelligent symbol attributes extraction from engineering drawings. Master's thesis, University of Turku, 2023.
- [12] G. Jocher. Ultralytics YOLOv5. <https://github.com/ultralytics/yolov5>, 2020.
- [13] M. T. Khan, L. Chen, Z. Yong, J. M. Tan, W. Feng, and S. K. Moon. From drawings to decisions: A hybrid vision-language framework for parsing 2d engineering drawings into structured manufacturing knowledge, 2025.
- [14] G. Kim, T. Hong, M. Yim, J. Nam, J. Park, J. Yim, W. Hwang, S. Yun, D. Han, and S. Park. Ocr-free document understanding transformer. In *Computer Vision – ECCV 2022*, pages 498–517, Cham, 2022. Springer Nature Switzerland.
- [15] Y.-H. Lin, Y.-H. Ting, Y.-C. Huang, K.-L. Cheng, and W.-R. Jong. Integration of deep learning for automatic recognition of 2d engineering drawings. *Machines*, 11(8), 2023.
- [16] Microsoft Corporation. Azure Blob Storage. <https://learn.microsoft.com/en-us/azure/storage/blobs/>, 2025. Accessed: 15 November 2025.
- [17] Microsoft Corporation. Azure machine learning documentation. <https://learn.microsoft.com/en-us/azure/machine-learning/?view=azureml-api-2>, 2025. Accessed: 11 November 2025.
- [18] Microsoft Corporation. Bandit policy. <https://learn.microsoft.com/en-us/python/api/azure-ai-ml/azure.ai.ml.sweep.banditpolicy?view=azure-python>, 2025. Accessed: 11 November 2025.
- [19] Microsoft Corporation. Azure AI Foundry. <https://learn.microsoft.com/en-us/azure/foundry/what-is-foundry>, 2026. Accessed: 20 February 2026.
- [20] Microsoft Corporation. Use the Azure OpenAI Responses API. <https://learn.microsoft.com/en-us/azure/foundry/openai/how-to/responses?tabs=python-key>, 2026. Accessed: 20 February 2026.
- [21] M. S. Mohd Yazed, E. F. Ahmad Shaubari, and M. H. Yap. A review of neural network approach on engineering drawing recognition and future directions. *JOIV : international journal on informatics visualization Online*, 7(4):2513–, 2023.
- [22] S. MORI, C. Y. SUEN, and K. YAMAMOTO. Historical review of OCR research and development: Optical character recognition. *Proceedings of the IEEE*, 80(7):1029–1058, 1992.

- [23] T. Nguyen, L. Pham, C. Nguyen, and V. Nguyen. Object detection and text recognition in large-scale technical drawings. In *10th International Conference on Pattern Recognition Applications and Methods*, pages 612–619, 2021.
- [24] OpenAI. gpt-5.0. <https://developers.openai.com/api/docs/models/gpt-5>, 2026. Accessed: 20 February 2026.
- [25] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, 2016.
- [26] R. Smith. An overview of the tesseract ocr engine. In *ICDAR '07: Proceedings of the Ninth International Conference on Document Analysis and Recognition*, pages 629–633, Washington, DC, USA, 2007. IEEE Computer Society.
- [27] J. V. Toro and M. Tarkian. Optimizing text recognition in mechanical drawings: A comprehensive approach. *Machines*, 13(3), 2025.
- [28] F. Ö. Ünel, B. O. Özkalayci, and C. Çiğla. The power of tiling for small object detection. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 582–591, 2019.
- [29] L. Wenying, W. Zhang, and L. Yan. An interactive example-driven approach to graphics recognition in engineering drawings. *International Journal of Document Analysis and Recognition (IJDAR)*, 9(1):13–29, 2007.
- [30] B. Xiao, H. Wu, W. Xu, X. Dai, H. Hu, Y. Lu, M. Zeng, C. Liu, and L. Yuan. Florence-2: Advancing a unified representation for a variety of vision tasks. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4818–4829, 2024.