Missing

1. Rich Picture of the situation at PSM, Context diagr.
2. Structured Glossary of terms.
3. Bibliography
4. "Related Work" (are there solutions to a similar problem?)
5. Architecture (a diagram) of your pipline
6. What exactly did the client request from you?
7. Your concept ML Pipeline ~~data~~ data aquisition (synth. data?)

ML Pipeline Overview

Contents

ToC it is generated

1 Overview

This project builds an end-to-end machine learning pipeline to evaluate technical blueprints by extracting dimensions, tolerances, and annotations—and then classifying them based on production feasibility. This document provides an overview of the pipeline's architecture and the sequential steps involved, as represented in the accompanying UML diagram (`ml_pipeline.puml`).

Your goal is to extract the tolerances.

Where is it?

Classification / assessment that the engineering drawing is feasible is done by an expert of PSM

2 Table of Contents

The sections in this document include:

3 Overview of the ML Pipeline

The ML pipeline is designed to process complex blueprint data in a sequential and modular manner. The process converts raw blueprint files into structured data, cleans and normalizes that data, extracts meaningful features, and ultimately utilizes multiple machine learning models to determine whether the specified tolerances are achievable in production. The key modules are:

- **Data Layer:** For file ingestion and parsing.
- **Extraction Module:** Converts blueprint images into structured, domain-specific data.
- **Data Processing and Preprocessing:** Cleans and normalizes the extracted data.
- **Feature Engineering:** Transforms normalized data into feature vectors.
- **Modeling:** Comprises several models:
  - **YOLO Detection Model:** Identifies key regions (bounding boxes) in blueprints.
  - **Symbol Classification Model:** Classifies symbols (e.g., arrows, icons) within cropped regions.
  - **OCR Model:** Extracts text from cropped regions.
  - **Rule-Based Feasibility Model:** Integrates model outputs to perform a feasibility analysis.

- **Deployment and Monitoring:** Packages and deploys the final model and monitors its performance.
- **Pipeline Orchestration:** Manages the sequential flow of tasks across the pipeline.

4 Pipeline Workflow Steps

Description of what you do on the conceptual level! No implementation details!

4.1 1. Data Ingestion and Parsing

- **DataIngestion**
  Method: `ingestData(sourcePath: str) → RawData`
  Function: Reads raw blueprint files (PDFs, CAD files) from specified sources.

- **FileParser (Base Class)**
  Method: `parse(filePath: str) → RawData`
  Function: Provides an interface for parsing files.

- **PDFParser & CADParser (Inherit from FileParser)**
  Methods: `parsePDF(filePath: str) → RawData` and `parseCAD(filePath: str) → RawData`
  Function: Handle format-specific parsing logic.

4.2 2. Blueprint Extraction

- **BlueprintExtractionModel**
  Method: `extractBlueprintData(filePath: str) → StructuredData`
  Function: Uses computer vision techniques (with OCR assistance) to extract dimensions, tolerances, and annotations from blueprints.

is this a Python-class (→ Pydantic?!)

- **OCRProcessor (Support Class)**
  Method: `processImage(imagePath: str) → TextData`
  Function: Provides basic OCR capabilities to assist in text extraction during blueprint extraction.

4.3 3. Data Preprocessing

- **DataPreprocessing**
  Method: `cleanData(rawData: RawData) → CleanData`
  Function: Removes noise and corrects errors in the structured data.

- **DataPreprocessing**
  Method: `normalizeData(cleanData: CleanData) → NormalizedData`
  Function: Standardizes units, formats, and scales data for consistent further processing.

Concepts not source code

4.4 4. Feature Engineering

- **FeatureEngineering**
  Method: `extractFeatures(data: NormalizedData) → FeatureVector`
  Function: Converts normalized data into a feature vector that captures essential parameters.

- **FeatureEngineering**
  Method: `aggregateFeatures(features: FeatureVector) → AggregatedData`
  Function: Aggregates individual features into summary statistics or higher-level representations.

4.5 5. Modeling

4.5.1 5A. Object Detection (YOLO)

- **YOLODetectionModel**
  Method: `trainYOLO(features: FeatureVector) → Model`
  Function: Trains the YOLO model to detect key regions (bounding boxes) in blueprint images.

- **YOLODetectionModel**
  Method: `predictYOLO(features: FeatureVector) → BoundingBoxes`
  Function: Predicts bounding boxes to identify regions of interest.

4.5.2 5B. Symbol Classification

- **SymbolClassificationModel**
  Method: `trainSymbolClassifier(croppedImage: Image) → Model`
  Function: Trains a model to classify symbols in cropped image regions.

- **SymbolClassificationModel**
  Method: `predictSymbols(croppedImage: Image) → Prediction`
  Function: Classifies symbols (e.g., arrows, icons) based on the cropped input.

4.5.3 5C. OCR for Text Extraction

- **OCRModel**
  Method: `trainOCR(imageData: ImageData) → Model`
  Function: Trains an OCR model specifically tuned to your blueprint data.

- **OCRModel**
  Method: `extractText(image: Image) → TextData`
  Function: Extracts precise textual information from cropped regions.

4.5.4 5D. Rule-Based Feasibility Analysis

- **RuleBasedFeasibilityModel**
  Method: `applyRules(yoloOutput: BoundingBoxes, symbolOutput: Prediction, ocrOutput: Tex`
  Function: Integrates outputs from YOLO, symbol classification, and OCR models to determine if production tolerances are feasible.

4.6 6. Deployment and Monitoring

- **ModelDeployment**
  Method: `packageModel(model: Model) → DeploymentPackage`
  Function: Packages the final feasibility model for deployment.

- **ModelDeployment**
  Method: `deployModel(deploymentPackage: DeploymentPackage) → Endpoint`
  Function: Deploys the model (e.g., as a REST API) for real-time usage.

- **Monitoring**
  Method: `monitorPerformance(endpoint: Endpoint) → PerformanceMetrics`
  Function: Continuously monitors the model's performance in production.

- **Monitoring**
  Method: `triggerRetraining() → None`
  Function: Initiates retraining if performance degrades.

4.7 7. Pipeline Orchestration

- **PipelineManager**
  Method: `runPipeline() → None`
  Function: Orchestrates the entire workflow—from data ingestion, extraction, preprocessing, feature engineering, and modeling, to deployment and monitoring—ensuring each step is executed in the correct sequence.

5 Why the Steps Are Organized This Way

- **Entry Point and Standardization:**
  Data ingestion and file parsing are the entry points, ensuring all raw data is collected and converted to a consistent format.

- **Specialized Extraction:**
  The blueprint extraction step converts visual, unstructured data into meaningful structured data. This is critical because subsequent processing relies on having accurate, domain-specific data.

- **Cleaning and Normalization:**
  Data preprocessing refines the extracted data by cleaning and normalizing it, so that the feature engineering step can effectively generate numerical representations.

- **Feature Engineering for ML Models:**
  Feature engineering transforms cleaned data into input vectors, which are necessary for training the object detection, symbol classification, and OCR models.

how does your input data look like? GIF-Pictures. Document your assumptions here.

- **Modeling Sequence:**
  - **YOLODetectionModel** runs first to locate areas of interest.
  - Outputs from YOLO then drive the symbol classification and OCR tasks, ensuring that only relevant image areas are analyzed.
  - The rule-based feasibility step integrates the outputs to make the final production feasibility determination.

+ Image labeles?

- **Deployment and Feedback:**
  Once the decision is made, the model is packaged and deployed, and its performance is continuously monitored to ensure long-term reliability.

- **Orchestration:**
  The PipelineManager oversees the entire workflow, ensuring each dependency is met and steps are executed in the correct order, maintaining the integrity and efficiency of the pipeline.

6 Conclusion

This ML pipeline is a comprehensive, modular, and sequential process designed to address the complexities of extracting and analyzing blueprint data for production feasibility. The modular design allows individual components to be developed, tested, and maintained independently, while the orchestration layer ensures smooth integration of all parts. This document, along with the accompanying UML diagram, provides a clear roadmap for both implementation and future development.

Workflow Summary and Data Documentation

How is this document related to the others?

Workflow Summary and Data Documentation

.1 Workflow So Far

he current project workflow follows a modular pipeline that includes:

1. **Synthetic Data Generation**: Using custom scripts, synthetic documents and forms were created to simulate real-world scenarios in the absence of production data.
2. **OCR Extraction Pipeline**: Text is extracted from synthetic images using an OCR engine (e.g., Tesseract or EasyOCR). This text is then cleaned and structured.
3. **Data Annotation**: Extracted OCR content is manually or semi-automatically annotated to create ground truth for model training.
4. **Classification Model Training**: A machine learning model is trained to classify extracted document content into predefined categories (e.g., invoice, order, delivery note).
5. **Evaluation and Reference Metrics**: Performance of the classification model is recorded to serve as a benchmark when real data becomes available.

Model Used and Rationale

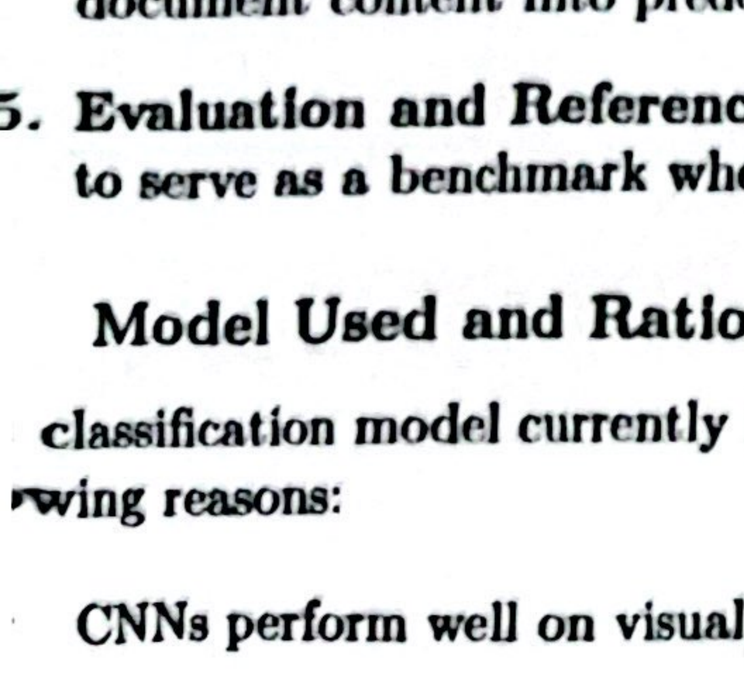
classification model currently in use is a Convolutional Neural Network (CNN), chosen for the wing reasons:

CNNs perform well on visual/textual patterns common in scanned document images.

The model architecture allows for end-to-end learning from preprocessed document images or OCR-extracted text representations.

It ca fine-tuned or extended for multi-label classification as required.

onsidered included traditional machine learning (e.g., SVM, Random Forest) to und on noisy OCR inputs. Future iterations may explore Transformer-ectures r LayoutLM) once more realistic datasets are available.



t Expected Results

- The OCR engine can extract structured and semi-structured text with reasonable accuracy, especially when synthetic image quality is high.
- Classification results from synthetic data training can be used as a baseline or reference for future comparison once real data becomes available.

These assumptions will be re-evaluated after real data is acquired.

1.4 Synthetic Data Documentation

Synthetic data was generated to simulate real business documents. Key details include:

are you using Faker or not?

- **Tools Used**: Python scripts using libraries like Faker for content, PIL/OpenCV for image generation, and LaTeX/HTML templates for layout.
- **Data Types**: Simulated invoices, delivery notes, and product orders, including varying layouts and fonts to introduce variability.

↑ drawings?!

- **Labeling Strategy**: Each synthetic document is labeled according to its document type for supervised classification.
- **Volume**: Approximately 500–1000 synthetic documents generated to cover various business cases and formats.
- **Storage Format**: Data stored as PNG/JPG images with associated labels in CSV/~~JSON~~ format for model training.

ok

The synthetic data allows for early-stage prototyping and serves as a placeholder until production data is available under NDA.

More details would be helpful.

?!

Notes to Entry ?

Note 3 to entry: A TED can be explicit or implicit. When indicated, an explicit TED is indicated by a rectangular frame including a value and sometimes an associated symbol, e.g. ∅ or R. On 3D models, explicit TEDs may be available by queries.

Note 4 to entry: An implicit TED is not indicated. An implicit TED is one of the following: 0 mm, 0°, 90°, 180°, 270° and the angular distance between equally spaced features on a complete circle.

Note 5 to entry: TEDs are not affected by individual or general specifications.

3.8 Theoretically Exact Feature (TEF)

⇐ What is the purpose of a TEF? Does our client need it?

Nominal feature with ideal shape, size, orientation and location, as applicable.

Note 1 to entry: A theoretically exact feature (TEF) can have any shape and can be defined by explicitly indicated theoretically exact dimensions (TEDs) or implicitly defined in CAD data.

Note 2 to entry: The theoretically exact location and orientation, if applicable, is relative to the indicated datum system for the specification of the corresponding actual feature.

Note 3 to entry: See also ISO 25378.

Example 1: The spherical surface shown in Figure 110 is a theoretically exact feature, with a defined spherical radius and a defined location and orientation relative to datum A.

Example 2: A virtual condition, e.g. a maximum material virtual condition (MMVC) according to ISO 2692, is a theoretically exact feature.

→ extracted from the norm?

3.9 United Feature

Compound integral feature which may or may not be continuous, considered as a single feature.

Note 1 to entry: A united feature can have a derived feature.

Note 2 to entry: The definition of a united feature is intentionally very broad to avoid excluding any useful applications. However, it is not intended that a united feature can be used to define something that is by nature several separate features. For example, building a united feature from two parallel, non-coaxial cylindrical features, or two parallel, non-coaxial rectangular tubes (each built from two perpendicular pairs of parallel planes) is not an intended use.

Example 1: A cylindrical feature defined from a set of arc features, such as the outside diameter of a spline, is an intended use of a united feature, see Figure 48.

Example 2: Two complete coaxial cylinders, which do not have the same nominal diameter, cannot be considered as a united feature.

Glossary of terms?

Where is your definition of the term:
- Tolerance ?
- Workpiece ?

1 - There are several kinds of

4 Basic Concepts

No plain copy out of the norm. You had to define the basic terms in your own words. ~~the~~ ~~title~~ ~~see~~ See Examples of glossaries in other books! A table or something similar.

4.1

Geometrical tolerances shall be specified in accordance with functional requirements. Manufacturing and inspection requirements can also influence geometrical tolerancing.

Note: Indicating geometrical tolerances does not necessarily imply the use of any particular method of production, measurement or gauging.

4.2

A geometrical tolerance applied to a feature defines the tolerance zone around the reference feature within which the toleranced feature shall be contained.

Note 1: In some cases, i.e. when using the characteristic parameter modifiers introduced in this document, see Figure 13, geometrical specifications can define characteristics instead of zones.

Note 2: All dimensions given in the figures in this document are in millimetres.

4.3

A feature is a specific portion of the workpiece, such as a point, a line or a surface; these features can be integral features (e.g. the external surface of a cylinder) or derived features (e.g. a median line or median surface). See ISO 17450-1.

What is defined in this norm?

4.4

→ Bibliography?

Depending on the characteristic to be specified and the manner in which it is specified, the tolerance zone is one of the following:

Where are the graphical symbols?

- the space within a circle;
- the space between two concentric circles;
- the space between two parallel circles on a conical surface;
- the space between two parallel circles of the same diameter;
- the space between two equidistant complex lines or two parallel straight lines;
- the space between two non-equidistant complex lines or two non-parallel straight lines;
- the space within a cylinder;
- the space between two coaxial cylinders;
- the space within a cone;
- the space within a single complex surface;
- the space between two equidistant complex surfaces or two parallel planes;

- the space within a sphere;

- the space between two non-equidistant complex surfaces or two non-parallel planes.

Note: The tolerance zone may be defined in the CAD model.

4.5

Unless a more restrictive indication is required, for example by an explanatory note, the toleranced feature may be of any form, orientation and/or location within this tolerance zone.

4.6

The specification applies to the whole extent of the considered feature unless otherwise specified. See Clauses 11 and 12. Currently, the detailed rules for partitioning (defining the boundary of the toleranced feature) are not elaborated in GPS standards. This leads to an ambiguity of specification.

Where are they?

4.7

Geometrical specifications which are assigned to features related to a datum(s) do not limit the form deviations of the datum feature(s) itself.

4.8

For functional reasons, one or more characteristics can be specified to define the geometrical deviations of a feature. Certain types of specifications, which limit the geometrical deviations of a toleranced feature, can also limit other types of deviations for the same feature.

- A location specification controls location deviation, orientation deviation and form deviation of the toleranced feature.

- An orientation specification controls orientation and form deviations of the toleranced feature but cannot control its location.

- A form specification controls only form deviations of the toleranced feature.

Project Goals – Feasibility Analysis: Object Detection

N°	Target Description	Success Criteria (Measurability)
Z1	Automate evaluation of mechanical drawings to reduce manual checks.	YOLO model detects relevant drawing area (bounding boxes). Manual check no longer required for initial analysis.
Z2	Detect the rectangle that contains technical info like values, tolerances, and part IDs.	Model identifies and localizes rectangles with ¿90% accuracy in test set.
Z3	Use OCR to extract relevant text/numbers from the detected rectangles.	OCR extracts text with ¿85% accuracy and maps to structured format (e.g., JSON).
Z4	Identify critical tolerances and feasibility issues based on extracted data and predefined rules.	System highlights values outside manufacturing specs automatically. Flagging works on ¿90% of cases.
Z5	Re-evaluate updated drawings automatically and highlight changes.	Updated parts are automatically compared to previous versions; changes are clearly marked.
Z6	Develop a user interface for reviewing detected data and manual corrections if needed.	Web interface displays detected values and allows edits. 100% of processed files are accessible via UI.
Z7	Log system decisions and create a summary report for each drawing processed.	Each processed drawing has a log file with extracted data, tolerances, and final feasibility decision.

Problem Statement – Automating Drawing Analysis for Tolerance Feasibility

~~Your Name~~

PSM Protech GmbH Project

May 3, 2025

Customer Problem (Engineering Perspective)

In the quotation phase at PSM Protech GmbH, customers submit detailed mechanical drawings containing dimensions, tolerances, GD&T symbols, and various callouts (commonly enclosed in ovals). Engineers must manually extract and interpret these symbols and values to assess manufacturability and process feasibility.

This manual extraction process is slow, error-prone, and highly repetitive — especially since a single drawing can contain ~~over 500 callouts and multiple versions.~~ Missing critical tolerances such as ±0.01 mm can lead to incorrect quotations, increased production costs, or delays. Automating this analysis can significantly improve speed, accuracy, and consistency in early design evaluation.

Great! ⇒ Export detected tolerances to xls. Evaluate tolerances manually ...

Technical Problem (Machine Learning/Computer Vision Perspective)

To automate the interpretation of these engineering drawings, we propose a multi-stage pipeline using AI and computer vision: ✓

- **Object Detection (YOLOv8):** Detect oval callouts and GD&T frames (feature control frames) in the drawing.
- **Image Classification:** Classify detected geometric tolerance ✓ symbols (e.g., flatness, roundness, perpendicularity) according to ISO 1101.
- **Optical Character Recognition (OCR):** Extract ✓ numerical tolerance values (e.g., ±0.05, Ø10.2), text annotations (e.g., "CC", "3x"), and datums (A, B, C).
- **Rule-Based Classification:** Use customer-defined feasibility rules to classify dimensions as either feasible (green) or critical/infeasible (red).
- **(Optional) Version Comparison:** Highlight changes between drawing revisions for re-review.

great!

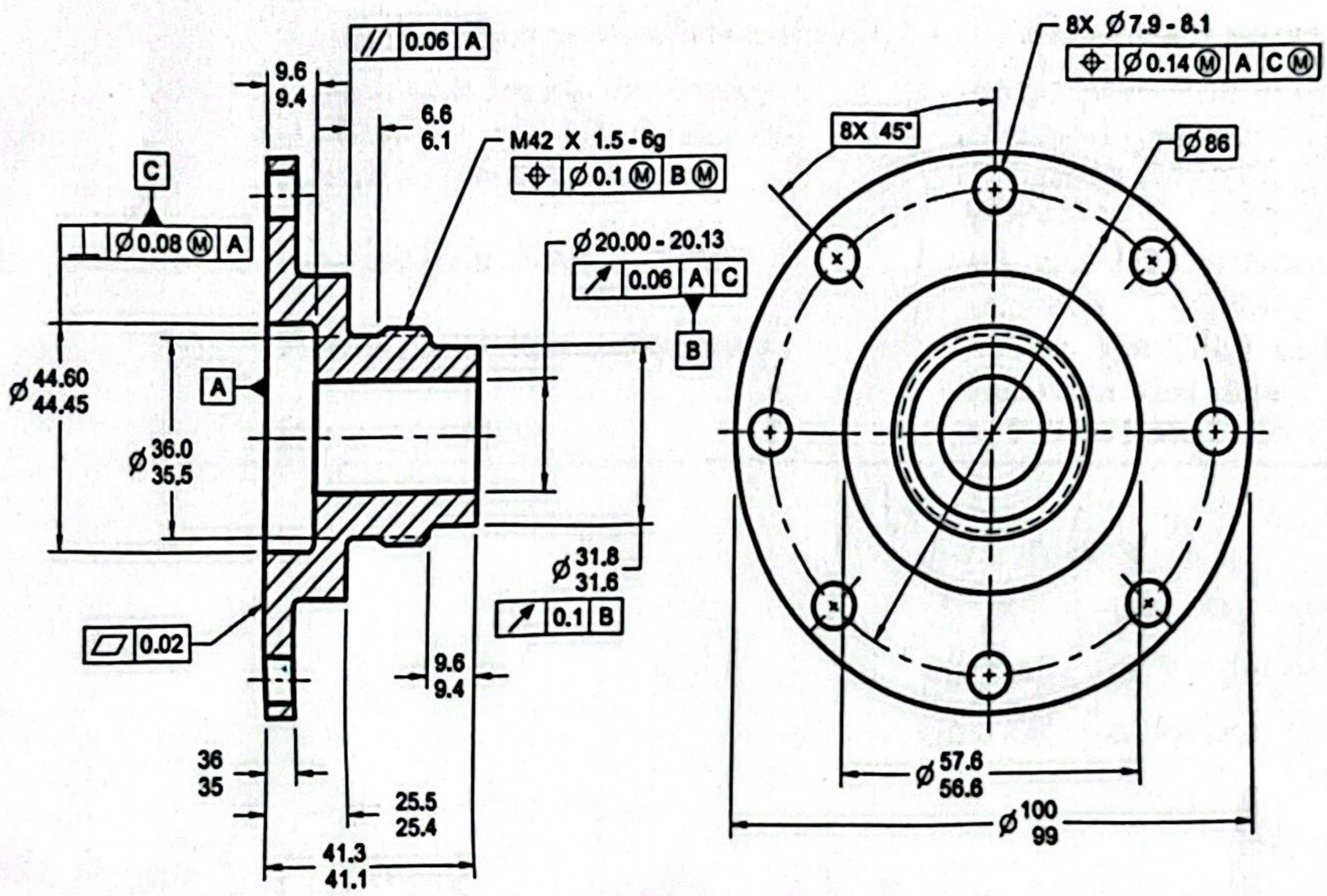


Figure 1: AI-based detection pipeline: YOLO identifies oval callouts and symbol blocks; image classification labels GD&T symbols; OCR extracts values for feasibility checking.

Suppl

Latex!

Table 1: Project Deliverables for Feasibility Analysis – Object Detection

N	Deliverable / Object / Output	How?
D1	Requirements & Specification Document	≈10 pp ~~Word~~/PDF detailing motivation, scope, tolerances, updates
D2	System Architecture & Design Document	≈8 pp ~~Word~~/PDF + ~~UML diagrams~~
D3	Data-Preprocessing Pipeline	Python scripts / Jupyter notebooks in Git repo + Docker image ← why?
D4	YOLO-based Object Detection Prototype	Ultralytics YOLO code in Git repo + Docker Compose why?
D5	Tolerance & Critical-Feature Detection Module	Python package with unit tests
D6	Highlight & Change-Reevaluation Module	Python scripts + example notebooks
D7	Integration & Deployment Guide	≈5 pp ~~Word~~/PDF + ~~Docker~~ image + ? (optional) Kubernetes manifest
D8	Interactive Demo Application	~~Python/Qt or~~ web UI, Docker + sample ~~PDFs~~ why? Request by Client?
D9	Demonstration Video	≈3 min MP4 screen-recording with voice-over Latex!
D10	Final Report & Recommendations	≈15 pp ~~Word~~/PDF including results, limitations, next steps
D11	Presentation Slides	≈15 slides PowerPoint ✓

deliver one integrated Solution with integrated python call

→ Readme.-File in Repo

OK

Architecture?
A diagram of the "distribution-Architecture" would be nice.

I am not quite sure,
that your client requested an that