Power-Line Component and Defect Detection – Technical Test Strategy

Author: Sébastien Dorgan
Date: December 2025

Context
This document was prepared as part of a technical test on automated inspection of medium-voltage overhead power lines.
The test material consists of: - An unlabeled dataset of 650 images of wooden and concrete poles. - The document “Test - Classification reseaux aeriens.pdf”, which defines the defect taxonomy and associated urgency levels.

From my understanding, the objective is not to fully annotate the provided dataset, but to: - Analyse the reference taxonomy and its applicability to vision-based inspection. - Select and harmonise public annotated datasets. - Propose a realistic end-to-end approach for component and defect detection using open-source tools and pretrained models, under tight time constraints.

Assumptions:

- the goal of the technical test is not to manually annotate the provided dataset.

- the goal of the technical test is not to benchmark state-of-the-art research NN models

Analysis of the provided test dataset

The test dataset contains 650 images of a medium-voltage overhead network with both wooden and concrete poles.

No labels or metadata are provided about the presence or absence of defects. In addition, many of the defects defined in the reference document (“Test - Classification reseaux aeriens.pdf”) are subtle or only indirectly visible. At the resolution and viewpoints available, it is often difficult to reliably identify and label defects by visual inspection alone while remaining consistent with the reference classification.

Constraints and practical considerations

The proposed strategy is driven by a set of practical constraints:

- **Time**: the test must be completed in roughly one week, which rules out full manual annotation of 650 images and extensive hyperparameter sweeps.
- **Data**: the test dataset is unlabeled and public datasets only partially match the target network (region, hardware types, voltage level).
- **Resources**: the solution should run on a single standard laptop GPU/CPU without requiring large-scale distributed training.
- **Tooling**: only open-source tools and publicly accessible datasets are assumed; no internal annotation teams or proprietary frameworks.
- **Scope**: the objective is to demonstrate a coherent, reproducible pipeline rather than to chase marginal performance gains.

These constraints justify the choice of YOLOv8s, reuse of public datasets, skipping class balancing, and relying on LLM-assisted pre-annotation only as an optional accelerator.

Analysis of defect types in “Test - Classification reseaux aeriens.pdf”

The reference document defines a detailed taxonomy of defects and assigns an urgency level to each situation. However, several defect types are either not directly detectable in vision or are insufficiently illustrated to be used as training classes.

Defects that are not (or not reliably) detectable from vision alone

From the document, some defects are explicitly defined through physical tests or measurements rather than pure visual inspection, for example:

- “Pourrissement au sol, cirons” (decay at ground level, wood-boring insects), which requires tests with hammer, screwdriver, auger, or Polux device.
- Risk of falling concrete (“risque chute de béton”) based on percussion tests (concrete sounding “hollow”) rather than purely on appearance.
- Certain cases of “armements inadaptés”, where the non-conformity is defined by reference to catalogues or mechanical calculations rather than a clearly visible shape.

These defects cannot be annotated consistently from images alone without the corresponding in-field measurements.

Defects with no or very limited visual examples in the document

Some defect categories are defined textually but have few or no visual examples in the PDF, for instance:

- “Pourrissement au sol, cirons”
- “Haubané provisoire” (temporary guying) for metallic supports and armaments
- “Risque chute de béton”
- “Armements inadaptés”

In section 4.2.4 on armaments, several different defect types (e.g. twisted, slipped, inappropriate, corroded) are grouped in a single descriptive line, while only two visual examples are provided for seven distinct defect situations. This lack of one-to-one correspondence between text and images makes it difficult to derive clear, separable visual classes that could be used as ground truth.

Choice of external annotated datasets

Based on the assumption that the test objective is not to manually annotate the provided 650 images, it is necessary to rely on public annotated datasets in order to:

- Detect the main components to be analysed (towers, insulators, conductors, etc.).
- Detect defects affecting those components, where possible.

For this exercise, I chose to use the Roboflow platform, which offers several relevant power-line datasets with existing annotations.

However, the available public datasets do not exactly match the type of network present in the test images:

- Most datasets depict North-American distribution networks, which differ in design and hardware from European networks.
- The largest datasets often focus on extra-high voltage transmission lines with large lattice steel pylons, whereas the test images mainly show medium-voltage lines on wooden or concrete poles.

These limitations affect both component detection (type and geometry of towers, insulators, conductors) and defect detection (type and appearance of damages).

Construction of the component-detection dataset

The selected Roboflow datasets use heterogeneous label taxonomies. The first step is therefore to harmonise the class definitions.

All selected datasets must be re-labelled so that their annotations map to a common set of component classes. For component detection, I retained the following classes:

- 0: `tower`
- 1: `insulator`
- 2: `spacer`
- 3: `damper`
- 4: `tower_plate`
- 5: `conductor`

To build this component-detection dataset:

- I reuse all selected images, including those originally annotated only for components and those annotated for specific defects.
- The objective is for the neural network to learn to detect components regardless of their condition, including cases where the defect strongly alters the visual appearance (e.g. a heavily damaged insulator).

The original datasets mix two types of annotations:

- Bounding boxes (object detection)
- Polygons or masks (instance/semantic segmentation)

All polygon annotations must therefore be converted to tight bounding boxes to obtain a consistent YOLO-style detection dataset.

Because the datasets come from different sources, class distributions are imbalanced. A proper dataset design would require class balancing (downsampling frequent classes, oversampling or augmenting rare ones). Given the limited time available for this technical test, I explicitly choose to skip this balancing step and work with the raw class distribution.

Construction of the defect-detection dataset

The same harmonisation approach is applied to defect-oriented datasets, but this time only images with explicit defect labels are retained.

The process is:

- Select only datasets where defects are annotated (e.g. broken insulators, damaged conductors, corroded hardware).

- Map the original defect labels to a reduced set of defect classes compatible with:
    - The visual criteria described in “Test - Classification reseaux aeriens.pdf”.
    - The actual content of the public datasets (for example, mainly defects on insulators, conductors, and hardware).
- Convert all annotations to bounding boxes, as for the component-detection dataset.

This produces a second dataset focused on defect detection, which can be used to fine-tune a model on top of the component-detection backbone, or as a separate model depending on the chosen architecture.

Choice of neural network architecture

For power-line inspection, more specialised and potentially more effective architectures than YOLO exist in the literature. For example:

- **DETR-family transformers** (DETR, Deformable DETR, DINO, DN-DETR) designed for high-quality detection with strong global reasoning.
- **RT-DETR** (Realtime DETR) and similar models that combine transformer heads with real-time performance.
- **Domain-specific detectors** for industrial inspection or aerial imagery (e.g. variants of Faster R-CNN or RetinaNet adapted to small objects and long, thin structures).

These models can outperform generic YOLO-style architectures in carefully engineered pipelines, but they require more integration work, more complex training recipes, and often custom annotation formats or codebases. For this technical test, the objective is to demonstrate a robust and reproducible end-to-end workflow rather than to benchmark state-of-the-art research models.

In this context, I chose **YOLOv8** as the main detection architecture because:

- It is **widely used and well-documented**, with mature tooling (Ultralytics) and an active ecosystem.
- It offers **strong baseline performance** for object detection, especially with pretrained weights and transfer learning.
- Many public datasets (including those on **Roboflow**) natively support **YOLOv8 export**, which removes the need for custom converters and format maintenance.

To stay within the time constraints of the test and keep training efficient, I use the **`yolov8s`** variant (small model):

- It has significantly fewer parameters than larger variants (`m`, `l`, `x`), reducing training time and GPU memory usage.
- It still provides adequate detection performance for this exercise, especially when fine-tuned on the harmonised datasets.

This choice offers a practical balance between **speed**, **implementation simplicity**, and **detection capability**, while keeping the pipeline compatible with a potential later migration to YOLOv11 or more specialised architectures if needed.

Cropping component-centered patches for defect detection

For defect detection, it is more effective to analyse crops centered on each component than the full image. Most defects are small relative to the image and require high effective resolution on the component itself. Full-image processing wastes capacity on background and distant objects while giving the defect only a few pixels.

A simple two-stage pipeline is therefore preferable:

1. **Stage 1 – Component detection**

    - Run a detector (e.g. YOLOv8) on the original images.
    - For each detection (tower, insulator, spacer, damper, tower_plate, conductor), extract its bounding box.

2. **Stage 2 – Defect detection/classification on crops**

    - Crop a patch around each bounding box, with a small padding to keep local context.
    - Resize the crop to a fixed size (e.g. 320×320 or 640×640).
    - Feed these patches to a dedicated model that predicts the presence and type of defect for that component.

This approach increases the apparent size of each component, reduces irrelevant background variability, and makes it easier for the defect model to focus on fine-grained visual cues while remaining compatible with the defect taxonomy in “Test - Classification reseaux aeriens.pdf”.

Fallback scenario: LLM-assisted annotation of the detection dataset

If the initial assumption (that the goal of the test is not to manually annotate the provided dataset) proves incorrect, a practical fallback would be to use large-model–assisted pre-annotation to accelerate the creation of a detection dataset.

To explore this option, I experimented with the **codex gpt-5 medium foundation model** on a small subset of the test images. The model was prompted to identify and localise key components (towers, insulators, conductors, etc.) and to output candidate bounding boxes and class labels suitable for object detection.

The results were **qualitatively interesting**: for the tested subset, the model was able to propose consistent annotations that were often close to what a human annotator would draw, especially for well-contrasted components. While these annotations still require human review and correction, they demonstrate that such a model could be used as a **pre-annotation tool** to:

- Generate initial bounding boxes and labels for the component-detection dataset.
- Reduce the manual effort to a faster “review and correction” pass instead of full annotation from scratch.

High-level pipeline overview

The overall strategy can be summarised as a two-stage detection pipeline built on harmonised public datasets:

1. Analyse the defect taxonomy from “Test - Classification reseaux aeriens.pdf” and identify which situations are usable from vision alone.
2. Select relevant public datasets on Roboflow and map their labels to a common set of component and defect classes.
3. Build and train a **Stage 1** YOLOv8s model for component detection on full images.
4. Use Stage 1 outputs to generate component-centered crops (with padding) on both public datasets and the test images.
5. Build and train a **Stage 2** model (detector or classifier) for defect detection on the component crops.
6. Run the full pipeline on the 650 test images and review qualitative results to assess feasibility and typical failure modes.

Evaluation strategy

Given the absence of ground-truth labels for the 650 test images, evaluation relies on a mix of quantitative and qualitative assessments:

- On public datasets, use standard train/validation splits and report classical detection metrics (mAP, precision, recall) per component and defect class.
- Monitor basic ablations (with/without cropping, different input sizes) to ensure that design choices are justified.
- On the unlabeled test set, perform a systematic **visual review** of detections on a representative subset of images, covering different viewpoints, backgrounds and hardware types.
- Document typical success cases (correct component and defect localisation) and typical errors (missed detections, false positives, domain-gap issues).

This evaluation is sufficient to demonstrate that the proposed pipeline is viable and to highlight where additional labelled data would bring the highest value.

Implementation plan (high-level)

Although the Python ecosystem has not yet fully caught up with this version, I chose to use **Python 3.14** because it marks an important turning point for the language:

- Significant improvements to error messages and the interactive console, which make debugging faster and more informative.
- A realistic path toward effective multithreading through an optional no-GIL runtime.
- The introduction of a built-in JIT / tiered interpreter, paving the way for future performance gains on compute-heavy workloads.

This choice is slightly ahead of the ecosystem, but it aligns the project with the direction in which modern Python is evolving.

The implementation can be organised as follow:

- Implement a small `RoboflowDownloader` utility to download and version public datasets in a reproducible way (roboflow API is not yet available in Python 3.14).
- Write a harmonisation script that remaps original labels to the chosen component and defect classes and converts polygons/masks to bounding boxes.

- Train the Stage 1 YOLOv8s component detector (using Ultralytics) and export trained weights.
- Implement a crop-generation script that applies Stage 1 to all images and saves component-centered patches with associated labels.
- Train the Stage 2 defect model on these patches, reusing pretrained weights where possible.
- Package inference scripts/notebooks to run the two-stage pipeline on the 650 test images and to visualise detections for qualitative review.

Despite the limited time, I aimed to produce professional-quality code supported by a consistent tooling setup, orchestrated via the `just` file. The implementation is organised as follows:

- Environment and dependencies are managed with `uv` to keep the Python toolchain and lockfile reproducible.
- Code style and static quality are enforced with `ruff` and `pyrefly`.
- Automated tests and coverage are handled with `pytest`.
- Security is checked with `osv-scanner`, aggregated in `just cqa` and `just ci` targets for local and CI-friendly runs.
- A single `just ci` command runs formatting, linting, type checking, tests, and security scanning, ensuring that every change can be validated quickly before reuse or extension.

Limitations and possible extensions

This strategy has several known limitations:

- It relies on public datasets that do not perfectly match the target network, which introduces a domain gap in appearance and hardware types.
- Only defects that are both visually defined in the reference document and present in public datasets are covered; other defect families remain unmodelled.
- Class imbalance and limited defect diversity are not fully addressed in this first iteration.

If the work were to continue beyond the test, several extensions would be natural:

- Collect a small, curated, hand-annotated dataset on the target network to reduce the domain gap.
- Experiment with more advanced or specialised detectors (YOLOv11, RT-DETR, transformer-based models) once the pipeline is stabilised.
- Add segmentation models for finer localisation of damaged areas on components.
- Industrialise LLM-assisted pre-annotation to bootstrap larger, higher-quality detection and defect datasets.