THEKER’s technical challenge

This is a technical challenge for you to show your acquired skills during your professional career.

If you’re completing this challenge means we think you could be a great addition to the team. Show us your worth!

Good luck 😊

1. Scenario for VLM:

Objective

Develop a **computer vision application** designed to run entirely on an **edge device** (no cloud access), capable of **detecting and localizing objects** in an image using **Vision-Language Models (VLMs)** or **LLMs with visual capabilities**. Unlike traditional approaches, **anchor-based detectors like YOLO (DINO) or Faster R-CNN are not allowed**.

Task Breakdown

- Work fully **on-device**, under limited memory and compute conditions (testing on the cloud local libraries, due to home-computer limitations, will be allowed)
- Accept **natural language input commands** (e.g., "Pick the pen") and return a **bounding box** over the correct object
- Be able to **detect and localize objects not seen during training**

Requirements

Your system must:

1. **Perform detection and localization** using:
    - Vision-Language Models (e.g., **Ollama**, **vLLM**, **CLIP**, **BLIP**, **SAM**, **TapNext**, etc.)
    - Without using anchor-based or pretrained detection models (YOLO, Faster R-CNN, SSD, etc.) as it must be as generalistic as posible
2. **Run locally on an edge device**:
    - No cloud services or APIs
    - Efficiency in **memory and processing** is key
3. **Handle unseen object categories**:
    - The system must **generalize to new objects** using textual prompts
    - Avoid reliance on fixed class labels or retraining
4. **Respond to natural language prompts**, such as:
    - "Take the scissors"
    - "Pick the pen"
5. **Return a bounding box** on the image indicating the object that matches the prompt

Deliverables

1. A **Python script or Notebook** implementing the full system
2. Example **input and output images**, showing the prompt and corresponding bounding box drawn on the image

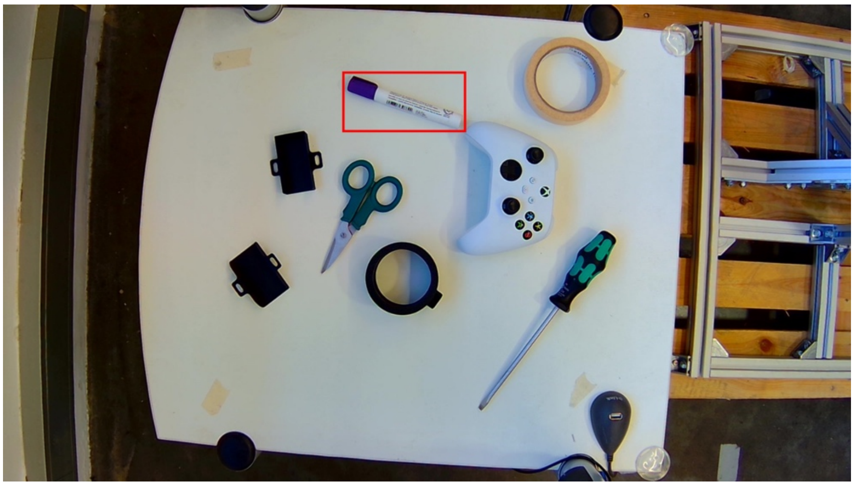
3. A **README or markdown summary** explaining:
    - Your detection and grounding strategy
    - Which models and tools were used
    - How the system handles unseen objects
    - How it runs efficiently on-device

Evaluation Criteria

- Correctness and precision of object localization based on the prompt
- Generalization to unseen or unlabeled objects
- System's ability to run efficiently on edge devices
- Clean and maintainable code

Example of use:
Input: "Pick the pen"

Output:



2. Challenge: Decode the Barcode

Objective

You are provided with a collection of **images containing barcodes**, captured under realistic conditions. Your mission is to **accurately extract the numeric code** embedded in each barcode.

This challenge focuses on building a robust and efficient **barcode decoding system**, emphasizing algorithmic precision over library-based shortcuts.

Task Breakdown

Your solution should cover two aspects:

1. **Barcode Detection** (optional, low priority):
   - Identify the position of the barcode in the image
2. **Barcode Decoding** (core of the challenge):
   - Develop a method to extract the **numerical code** from the detected barcode
   - You must implement your **own decoding algorithm** (see constraints below)

Constraints

- You **must not use** ready-made barcode decoders (although you can use them as a benchmark) such as:
  - Pyzbar, Zxing, Dynamsoft, or similar libraries
- You **may use**:
  - **Deep Learning approaches** (e.g., YOLO, CNNs, OCR architectures, etc.)
  - **Classic Computer Vision techniques** (e.g., thresholding, binarization, signal processing, morphology, etc.)
- You are free to work in **Python**, and can use libraries such as **OpenCV**, **NumPy**, **Pytorch**, **Scikit-image**, etc.

Deliverables

1. A **Python script or Jupyter Notebook** that processes all images and decodes the barcode from each one
2. A **text or CSV file** containing the decoded numbers, indexed by image filename
3. Example output images with optional visualizations (e.g., detection boxes, signal profiles, etc.)
4. A short **README** or markdown cell describing:
   - Your decoding strategy
   - Key techniques used

    - Limitations and potential improvements

Evaluation Criteria

- **Accuracy** of the decoded numbers
- **Speed** of your decoding approach
- **Code structure and clarity**
- **Robustness** to image imperfections (rotation, blur, lighting, etc.)

3. Normal angle package picking

Objective

Given an input image of **scattered packages** (with various shapes, sizes, colors, and textures) lying on the ground, your task is to:

- **Detect the optimal picking surface** of each package
- **Estimate the surface normal** vector for each detected picking area
- **Visualize the result** by drawing an arrow over the image that represents the estimated **normal vector** at the optimal picking point (drawing the optimal picking surface will be an added value to the evaluation)

Requirements

Your solution should:

1. **Process a real or synthetic RGB image** (a sample batch will be provide)
2. **Identify multiple objects/packages** in the scene
3. For each package:
    - Determine the **optimal surface point** or region for a robotic pick (e.g., top face, flat region, etc.)
    - Compute the **surface normal vector** at that point
    - Represent the normal as a **2D or 3D arrow overlay** on the image
4. The code must be written in **Python**
    - You may use **OpenCV**, **NumPy**, **Pytorch**, **Open3D**, or any open-source library of your choice

- Solutions that make use of depth data (if simulated or provided) are a plus, but **not mandatory**

Deliverables

1. A **Python script or Jupyter Notebook** that performs the task
2. One or more **example output images** with normals visualized
3. A short **README** or markdown cell that explains:
    - Your approach
    - Any assumptions made
    - Libraries used
    - Limitations or potential improvements

Evaluation Criteria

- Correctness of the surface normal estimation
- Clarity and structure of your code
- Quality of the visualization
- Simplicity and effectiveness of your approach
- Bonus: handling partial occlusions or noisy images gracefully

Optional Bonus

- Simulate or integrate **depth estimation** (from stereo or monocular techniques)
- Return the **(x, y, angle)** in degrees that the robot should use to perform the pick