

# Gold Nanoparticle Counter

## User Manual

Version 1.1

### Authors:

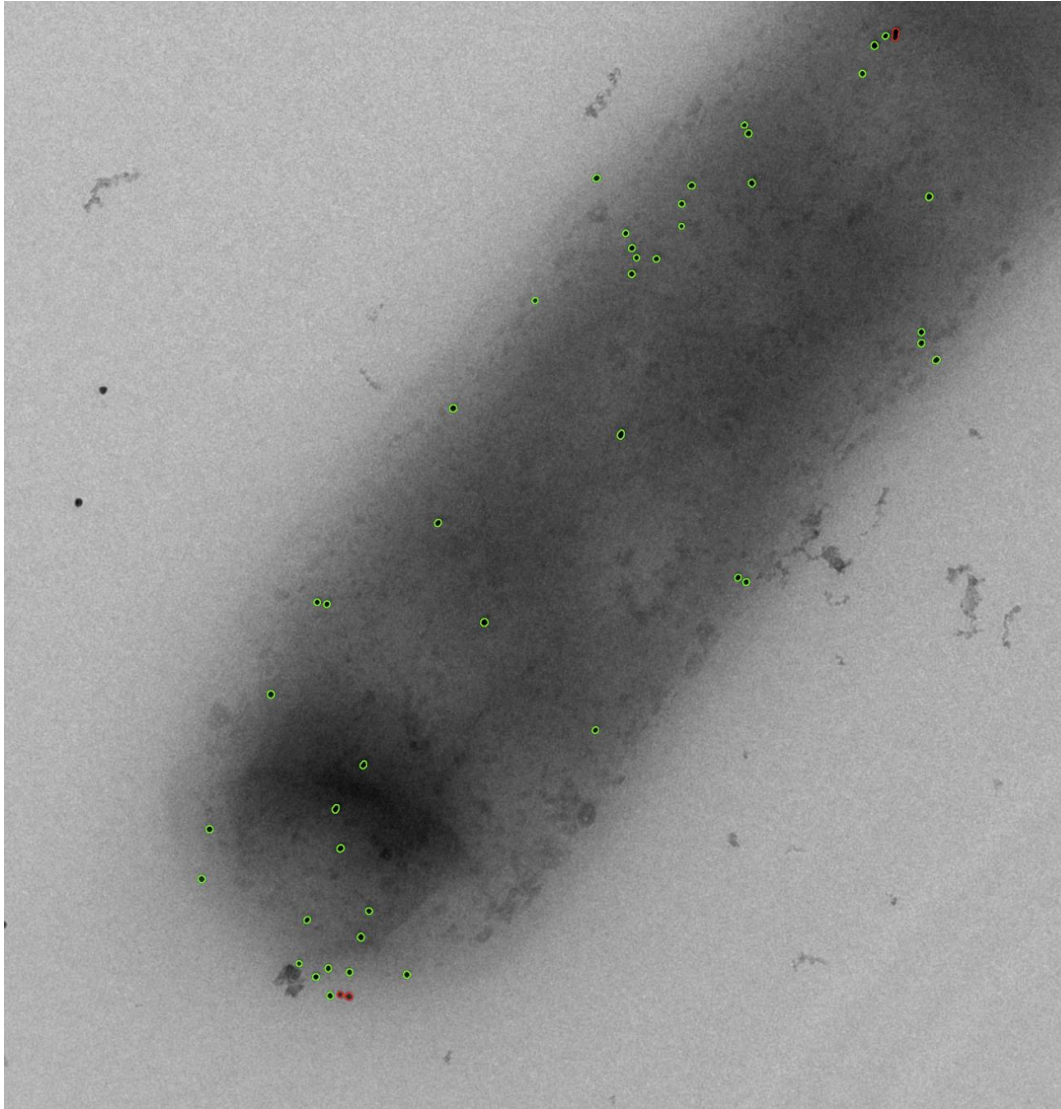
Bram Koobs ([b.l.koobs@st.hanze.nl](mailto:b.l.koobs@st.hanze.nl))

Michiel Meeuwisse ([michiel.meeuwisse@gmail.com](mailto:michiel.meeuwisse@gmail.com))

### Institutions

Hanze University of Applied Sciences

University of Amsterdam



## Table of Contents

1. Title page
2. Introduction
3. How the pipeline works
4. Using the application
  - Getting started
  - Uploading and processing image
  - Viewing and managing results
  - Tagging system
  - Adjusting dot count
  - Exporting Data
  - Settings

- Troubleshooting
- 5. Technical specifications
  - Tools and libraries
  - Data Representation
  - Folder Structure
- 6. How it works
  - Dot detection pipeline
  - Frontend
  - API
- 7. Discussion and conclusion
- 8. Possible future improvements
- 9. Credits and references

# Introduction:

## Context and Goals

The Gold Nanoparticle Counter is a software tool designed to automate the detection and counting of gold nanoparticles in transmission electron microscopy (TEM) images of tuberculosis cells. In biomedical research, gold nanoparticles are often used as labels to visualize specific proteins. By attaching gold nanoparticles to antibodies or other protein markers, researchers can track the presence and distribution of these proteins on or within cells. When imaged with an electron microscope, these nanoparticles appear as distinct dark spots, making it easier to count the proteins.

Traditionally, counting these nanoparticles was a manual and time-consuming process, prone to human error and subjectivity. Our tool aims to streamline this workflow by providing a robust, reproducible, and user-friendly solution for automated counting. This not only saves time but also improves the consistency and scalability of data analysis in research projects involving protein labeling.

The project was commissioned by the University of Amsterdam and developed in collaboration with the Hanze University of Applied Sciences. The primary goal was to create an accessible application that could be used by researchers with minimal programming experience, while still offering advanced customization for power users. The tool is intended to support studies in microbiology, cell biology, and related fields where gold nanoparticle labeling is used.

## Scientific Background

Labeling proteins with gold nanoparticles is a well-established technique in cell biology and immunoelectron microscopy. The gold particles, typically 5–20 nm in diameter, are conjugated to antibodies or other binding molecules. When these conjugates bind to their target proteins, the gold particles serve as electron-dense markers that are easily visualized in TEM images. This approach enables researchers to localize and quantify proteins at the ultrastructural level.

## **Project Objectives**

The main objectives of this project were:

- To develop an automated tool for detecting and counting gold nanoparticles in TEM images.
- To provide an intuitive graphical interface for researchers to upload images, adjust detection parameters, and manage results.
- To make sure that the results are easy to manage and sort
- To ensure reproducibility and scalability in nanoparticle quantification.

# How the Pipeline Works

## Overview of the Detection Pipeline

The Gold Nanoparticle Counter processes each image through a series of well-defined steps to achieve accurate and reproducible nanoparticle counting. The pipeline is designed to be robust to variations in image quality and adaptable to different experimental conditions.

### 1. Image and Configuration Loading

Upon receiving an image, the software loads user-defined or default configuration settings. These settings include parameters such as minimum dot area, blur size, and thresholds for distinguishing single dots from clusters. If enabled, the tool can use a machine learning model to predict optimal settings based on image properties.

### 2. Cell Detection

The first analytical step is to identify the main cell in the image. The image is blurred and thresholded to enhance contrast, and contours are detected. The largest or most relevant contour is selected as the cell boundary. This step ensures that only nanoparticles within the cell are counted, reducing background noise.

### 3. Image Preprocessing

The region of interest (the cell) is further processed to enhance the visibility of gold nanoparticles. The image is converted to grayscale, blurred, and thresholded to highlight dark spots corresponding to nanoparticles. Additional filtering steps may be applied to reduce noise and improve detection accuracy.

### 4. Dot Detection and Classification

Contours are detected in the preprocessed image. Each contour is analyzed for area and circularity. Small, round contours are classified as single nanoparticles, while larger or less circular contours are considered clusters. The software estimates the number of nanoparticles in each cluster based on its area and the average size of single dots.

## **5. False Positive Filtering**

To minimize false positives, the intensity of each detected region is compared to its surroundings. Only regions that are significantly darker than their local background are retained as valid detections.

## **6. Output Generation**

The results are visualized by overlaying detected dots and clusters on the original image. The annotated image and a corresponding JSON file containing detailed results (counts, area, tags, etc.) are saved to the output directory.

## **7. User Interaction and Adjustment**

Users can review the results in the application, manually adjust dot counts, assign tags, and export data for further analysis.

# Using the application

## Application Overview

Upon launching the Gold Nanoparticle Counter, users are greeted with a web-based interface featuring three main tabs: the main counter, application settings, and credits. The interface is designed for ease of use, with clear navigation and contextual help.

## Getting Started


Step 1, install python:

The version of python that was used to create this application is 3.11.8 (newer version will most likely work, but these have not been tested). To install version 3.11.8, you can download the installer at this link:

<https://www.python.org/downloads/release/python-3118/>

Download the python installer by navigating to the bottom of the page and download the installer that says recommended behind it.

### Files

| Version   | Operating System | Description   | MD5 Sum                          | File Size | GPG                 | Sigstore                  |
|---|------------------|---|----------------------------------|-----------|---------------------|---------------------------|
| <a href="#">Gzipped source tarball</a>              | Source release   |   | 7fb0bfaa2f6aae4aadcd51abe957825  | 25.3 MB   | <a href="#">SIG</a> | <a href="#">.sigstore</a> |
| <a href="#">XZ compressed source tarball</a>        | Source release   |   | b353b8433e560e1af2b130f56dfbd973 | 19.1 MB   | <a href="#">SIG</a> | <a href="#">.sigstore</a> |
| <a href="#">macOS 64-bit universal2 installer</a>   | macOS            | for macOS 10.9 and later  | 0903e86fd2c61ef761c94cb226e9e72e | 42.7 MB   | <a href="#">SIG</a> | <a href="#">.sigstore</a> |
| <b>Windows installer (64-bit)</b>                   | Windows          | Recommended  | 77d17044fd0de05e6f2cf4f90e87a0a2 | 24.9 MB   | <a href="#">SIG</a> | <a href="#">.sigstore</a> |
| <a href="#">Windows installer (32-bit)</a>          | Windows          |   | 45d4b29f26ca02b1ccf13451ea136654 | 23.7 MB   | <a href="#">SIG</a> | <a href="#">.sigstore</a> |
| <a href="#">Windows installer (ARM64)</a>           | Windows          | Experimental  | ae1b38fa57409d9a0088a031f59ba625 | 24.2 MB   | <a href="#">SIG</a> | <a href="#">.sigstore</a> |
| <a href="#">Windows embeddable package (64-bit)</a> | Windows          |   | 9199879fbad4884ed93ddf77e8764920 | 10.7 MB   | <a href="#">SIG</a> | <a href="#">.sigstore</a> |
| <a href="#">Windows embeddable package (32-bit)</a> | Windows          |   | 104bf63ef10c06298024a61676a11754 | 9.6 MB    | <a href="#">SIG</a> | <a href="#">.sigstore</a> |
| <a href="#">Windows embeddable package (ARM64)</a>  | Windows          |   | 6b989558c662f877e2e707d640673877 | 10.0 MB   | <a href="#">SIG</a> | <a href="#">.sigstore</a> |

When you have downloaded the installer, you can open it up and follow all the steps until Python is installed.

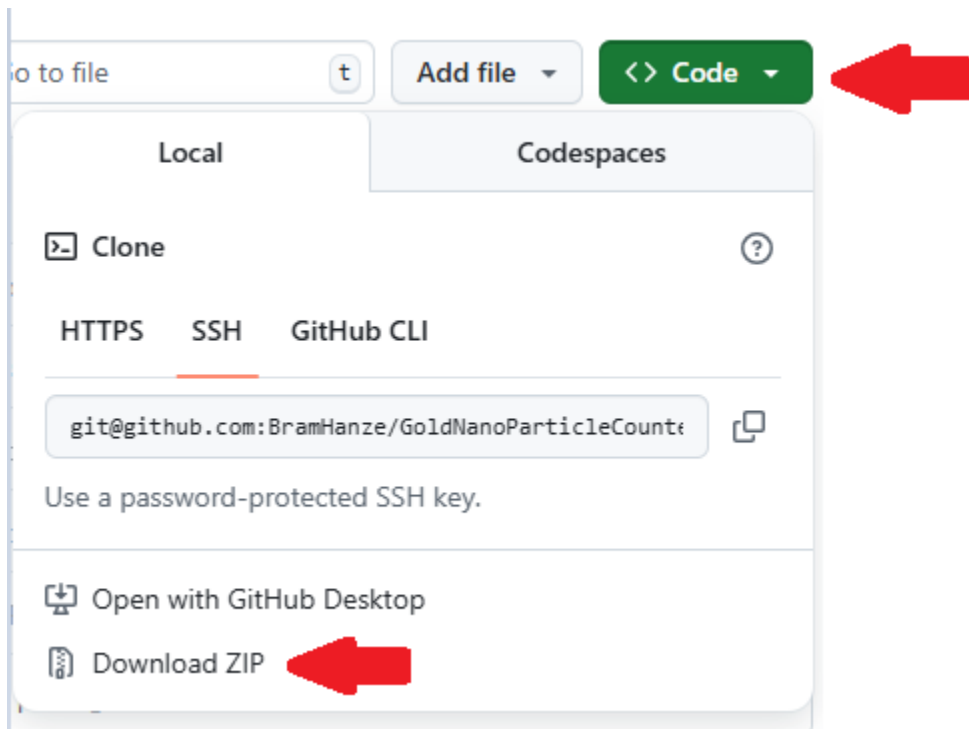
Step 2, Download the Gold nanoparticle detector:

You can download the gold nanoparticle detector from GitHub by going to this link:

[https://github.com/BramHanze/GoldNanoParticleCounter\\_TC](https://github.com/BramHanze/GoldNanoParticleCounter_TC)



Once you are on this page you first click the green code button and then the Download ZIP button.



You can unpack this ZIP file by using software like 7ZIP which you can download here:

<https://www.7-zip.org/download.html>

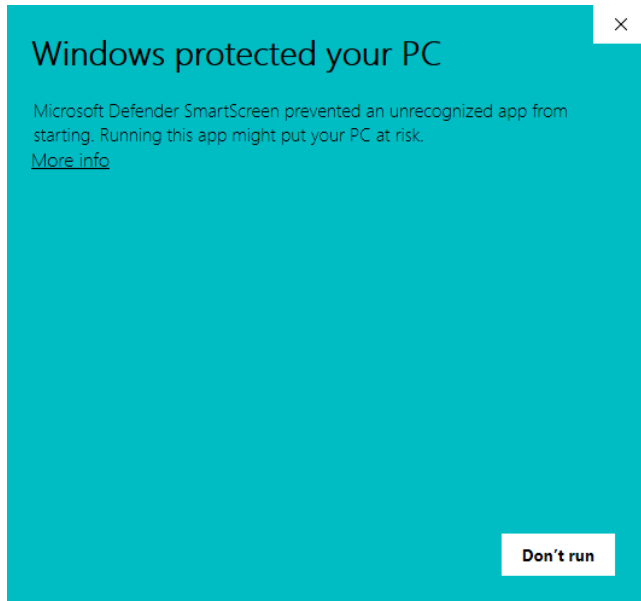
Step 3, run the application

To run the application, double click on 'run.bat'

| Name             | Type               | Size |
|------------------|--------------------|------|
| Backend          | File folder        |      |
| Documentatie     | File folder        |      |
| features         | File folder        |      |
| Frontend         | File folder        |      |
| __init__.py      | Python File        | 0 KB |
| README.md        | Markdown Source... | 1 KB |
| requirements.txt | Text Document      | 2 KB |
| run.bat          | Windows Batch File | 1 KB |

All dependencies will automatically be installed if not already installed.

If windows flags the application as an unrecognized app, click on [More info](#), and then on 'Run Anyway'. This warning should only show up the first time the tool is executed.

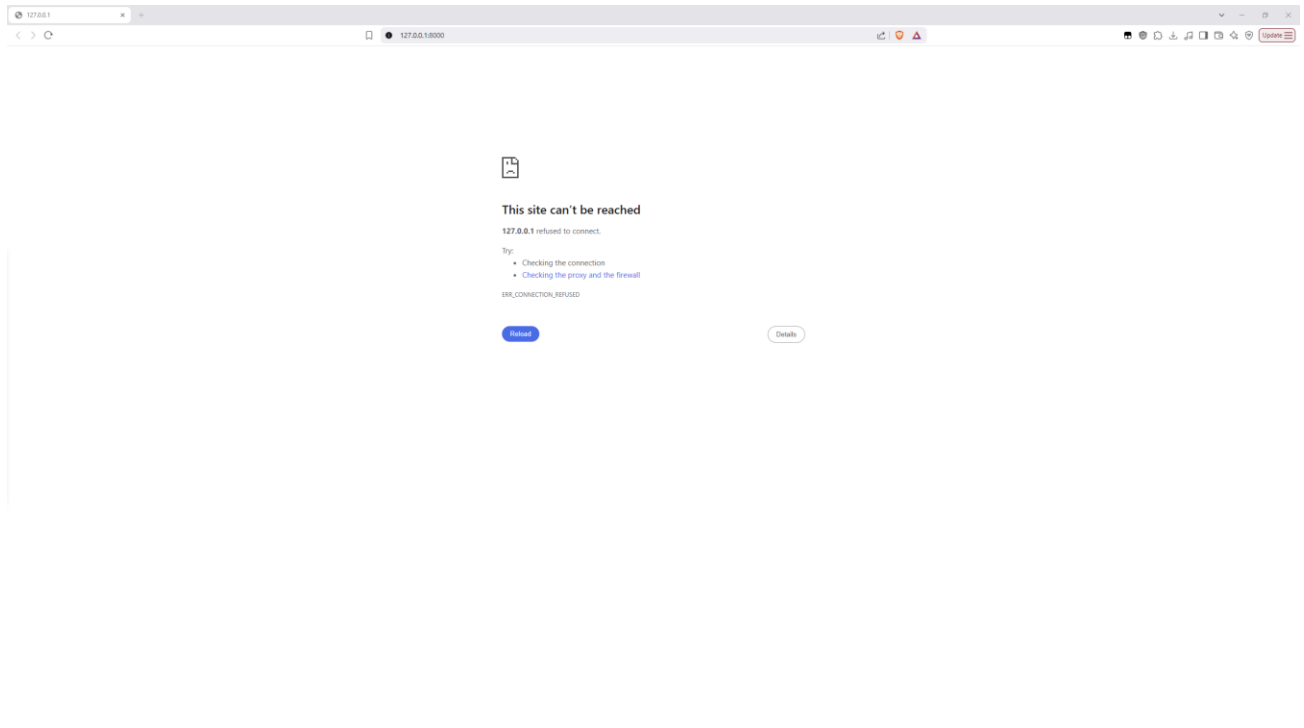


A command prompt / terminal will open, which will start downloading all needed dependencies after a few seconds if no '.venv' is found.

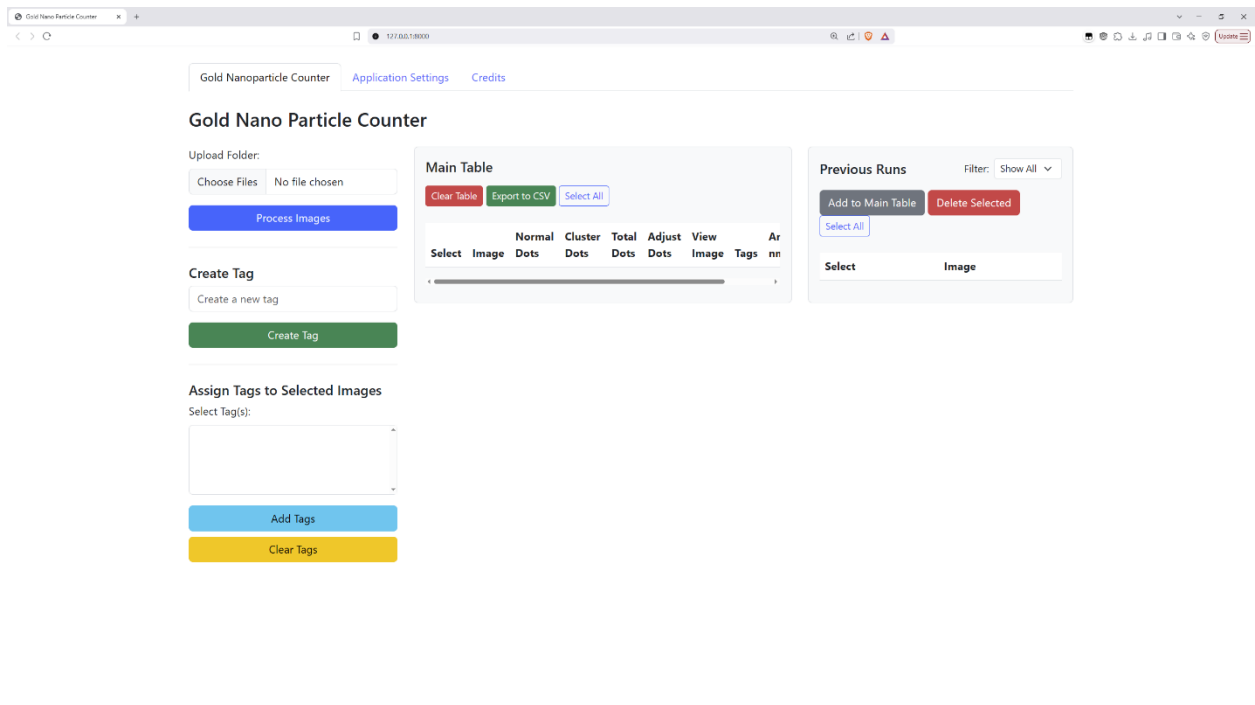
```
C:\Windows\system32\cmd.exe

Using cached shellingham-1.5.4-py2.py3-none-any.whl (9.8 kB)
Using cached six-1.17.0-py2.py3-none-any.whl (11 kB)
Using cached sniffio-1.3.1-py3-none-any.whl (10 kB)
Using cached stack_data-0.6.3-py3-none-any.whl (24 kB)
Using cached sympy-1.14.0-py3-none-any.whl (6.3 MB)
Using cached threadpoolctl-3.6.0-py3-none-any.whl (18 kB)
Using cached tifffile-2025.6.11-py3-none-any.whl (230 kB)
Using cached torch-2.7.1-cp313-cp313-win_amd64.whl (216.1 MB)
Using cached torchvision-0.22.1-cp313-cp313-win_amd64.whl (1.7 MB)
Using cached tornado-6.5.1-cp39-abi3-win_amd64.whl (444 kB)
Using cached traitlets-5.14.3-py3-none-any.whl (85 kB)
Using cached typer-0.16.0-py3-none-any.whl (46 kB)
Using cached typing_inspection-0.4.1-py3-none-any.whl (14 kB)
Using cached typing_extensions-4.14.0-py3-none-any.whl (43 kB)
Using cached tzdata-2025.2-py2.py3-none-any.whl (347 kB)
Using cached uvicorn-0.35.0-py3-none-any.whl (66 kB)
Using cached watchfiles-1.1.0-cp313-cp313-win_amd64.whl (292 kB)
Using cached wcwidth-0.2.13-py2.py3-none-any.whl (34 kB)
Using cached websockets-15.0.1-cp313-cp313-win_amd64.whl (176 kB)
Installing collected packages: wcwidth, pywin32, pytz, python-bidi, pyclicker, pure_eval, pathlib, mpmath, websockets, tzdata, typing_extensions, traitlets, tornado, threadpoolctl, sympy, sniffio, six, shellingham, setuptools, pyzmq, PyYAML, python-multipart, python-dotenv, pyparsing, Pygments, psutil, prompt_toolkit, platformdirs, pillow, parso, packaging, numpy, ninja, networkx, nest-asyncio, mdurl, MarkupSafe, kiwisolver, joblib, idna, httptools, h11, fsspec, fonttools, filelock, executing, dnspython, decorator, debugpy, cyclical, colorama, certifi, asttokens, annotated-types, typing-inspection, tifffile, stack-data, shapely, scipy, python-dateutil, pydantic_core, opencv-python-headless, opencv-python, matplotlib-inline, markdown-it-py, lazy_loader, jupyter_core, Jinja2, jedi, ipython_pygments_lexers, imageio, httpcore, email_validator, contourpy, comm, click, anyio, watchfiles, uvicorn, torch, starlette, scikit-learn, scikit-image, rich, pydantic, pandas, matplotlib, jupyter_client, ipython, httpx, typer, torchvision, seaborn, rich-toolkit, ipykernel, fastapi, fastapi-cli, easyocr
```

When it is done downloading all dependencies, a page will open in your browser.



After a few seconds the tool will be displayed on this page:



The command prompt will stay open until it is manually closed, closing it will (safely) shut down the tool.

If you want to create a moveable shortcut, right click on the run.bat and select 'Create Shortcut'. This shortcut will have the same name, but on the icon it will have a small blue arrow. This shortcut can be moved to any folder including the background/desktop.

| Name             | Type               | Size |
|------------------|--------------------|------|
| .venv            | File folder        |      |
| __pycache__      | File folder        |      |
| Backend          | File folder        |      |
| Documentatie     | File folder        |      |
| features         | File folder        |      |
| Frontend         | File folder        |      |
| output           | File folder        |      |
| __init__.py      | Python File        | 0 KB |
| README.md        | Markdown Source... | 1 KB |
| requirements.txt | Text Document      | 2 KB |
| run.bat          | Windows Batch File | 1 KB |
| run.bat          | Shortcut           | 2 KB |

## **Uploading and Processing Images**

To analyze images, click the "Upload Folder" button and select a folder containing your microscopy images (supported input: .tif). After selecting the folder, click "Process Images." A loading bar will appear while the backend processes the images. If "Only Run New Images" is enabled in the settings, previously processed images will be skipped.

Supported image output formats: .jpg, .png, .jpeg (these can be selected in 'Application Settings')

## **Viewing and Managing Results**

The main table displays the results for the currently selected images, including image name, counts of normal and cluster dots, total dots, manual adjustments, tags, cell area, and dot density. You can select or unselect all rows for bulk actions, clear the table, or view processed images in detail. The "Previous Runs" panel lists all previously processed images, with options to filter by tag, add results to the main table, or delete selected results.

## **Tagging System**

Tags can be created and assigned to images for better organization. To create a tag, enter a name and click "Create Tag." To assign tags, select images in the main table, choose tags from the dropdown, and click "Add Tags." Tags can be cleared in a similar manner. Filtering by tag is available in the "Previous Runs" panel.

## **Adjusting Dot Counts**

If the automated count needs correction, you can manually adjust the dot count for any image in the main table. The adjusted value is saved automatically, and the total dot count and density are updated in real time.

## **Exporting Data**

Click "Export to CSV" to download the current results as a CSV file. The export includes all relevant information, making it easy to analyze results in external tools such as Excel.

# Settings

## General Settings

### Predict Best Settings

When enabled, the application uses a machine learning model to automatically predict optimal detection parameters (such as minimum dot area and blur) based on the properties of each image. If disabled, the manually entered settings are used.

### Minimum Dot Area

Sets the smallest area (in pixels) that a detected contour must have to be considered a nanoparticle. Increasing this value can help filter out noise and very small artifacts; while decreasing it allows detection of smaller particles.

### Dot Blur

Specifies the size of the median blur kernel applied to the image before dot detection. Higher values can help reduce noise but may also blur out small dots.

### Only Run New Images

If enabled, the application will skip images that have already been processed, preventing duplicate analysis.

## Advanced Settings

### Single Dot Circularity Threshold

Defines the minimum circularity required for a contour to be classified as a single dot. Circularity is calculated as  $(4\pi \times \frac{\text{Area}}{\text{Perimeter}^2})$ , with values closer to 1 indicating a perfect circle.

### Cluster Circularity Threshold

Sets the maximum circularity for a contour to be considered a cluster. Lower values allow for detection of more irregularly shaped clusters.

### Dynamic Cluster Circularity

When enabled, the circularity threshold for clusters is adjusted dynamically based on the estimated number of dots in the cluster, allowing for more flexible cluster detection.

### Cluster Area Factor

Determines how much larger a contour must be (relative to the average single dot area) to be considered a cluster. This helps distinguish between single dots and groups of overlapping dots.

### **Intensity Threshold**

Sets the minimum intensity difference between a detected dot and its surrounding area. This helps filter out false positives by ensuring that only sufficiently dark regions are counted as dots.

## **Output Settings**

### **Output Directory**

Specifies the folder where processed images and result files (JSON) will be saved.

### **Output Image Type**

Sets the file format for saving processed images. Supported formats include .jpg, .png, and .jpeg.

### **Dot Colour**

Determines the color used to mark detected single dots on the output image.

### **Cluster Dots Colour**

Sets the color used to mark detected clusters on the output image.

### **Show Cell Outline**

If enabled, the outline of the detected cell will be drawn on the output image.

### **Cell Outline Colour**

Specifies the color used for drawing the cell outline.

## **Cell Detection & Dot Detection**

### **Cell Detection Parameters**

These include settings for the blur and thresholding steps used to identify the cell region in the image. Adjusting these can help the algorithm better segment the cell from the background, especially in images with varying contrast.

### **Dot Detection Parameters**

Includes settings for the blur, threshold, and area filtering steps used to identify nanoparticles within the cell. Fine-tuning these parameters can improve detection accuracy for different image qualities and experimental conditions.

## Reset Buttons

### Reset to Default

Most input fields (except checkboxes) have a "Reset" button next to them. Clicking this button will revert the field to its default value as specified in the default configuration.

### Troubleshooting

If images are not processed, ensure they have not already been processed and that the input format is correct. If settings are not saved, make sure to expand advanced settings before submitting. If tags or adjustments are not updating, try refreshing the page or restarting the backend server.

### File Structure

If everything is correct the tool should have the following file structure

#### *Frontend/*

- **client.html**: Main web interface.
- **Settings\_content.html**: Settings page web interface
- **main.py**: FastAPI backend serving the UI and API.
- **config.yml, config\_default.yml**: Settings files.
- **output/**: Processed images and JSON results.
  - **Tags/**: The folder containing the tags file
    - **tags.json**: the file containing all the tags

#### *Backend/*

- **blackdotdetector.py**: Core detection logic.
- **scale\_finder.py**: Automatic scale detection using easyOCR
- **model.pkl**: ML model for parameter prediction.

### Notes

- All results (images, JSON, tags) are stored in the configured output directory.
- Tags are managed (and currently only able to be deleted) in Output/Tags/tags.json.
- Each processed image has a corresponding .json file with all result data and tags.



## How it works

### Dot detection pipeline

The BlackDotDetector begins by loading its configuration and the target image. It sets detection parameters such as minimum dot area and blur size using a model to predict the correct settings if this is enabled. If this is not enabled, it will use the selected settings. The pipeline starts with cell detection, where the image is blurred and thresholded to isolate the main cell region. Contours are found and filtered to identify the cell, which is expected to be a single, prominent contour.

Once the cell is detected, the image undergoes preprocessing: it is blurred, converted to a usable grayscale, and thresholded to highlight dark spots that may represent dots. Another blur and binary thresholding step produces a clean binary image for dot detection. The algorithm then finds all contours in this processed image and checks whether they are inside or near the detected cell, keeping only those that are relevant.

For each remaining contour, the pipeline filters by area and calculates circularity to distinguish between single dots and clusters. Single dots are typically small and round, while clusters are larger and less circular. The algorithm may further filter out false positives by comparing the intensity of the detected region to its surroundings. For clusters, it estimates the number of dots based on the area and average dot size.

Finally, the results are drawn on the image, including detected dots, clusters, and optionally the cell outline. The annotated image and a JSON file with the counts of normal dots, cluster dots, and total dots are saved. Optionally, the pipeline can also print statistics about the detected dots for further analysis. The run method orchestrates all these steps, ensuring the image is processed from start to finish in a reproducible way.

### Frontend

The frontend is a web application built with HTML, JavaScript, and Bootstrap, served by FastAPI. When a user opens the app in their browser, they are presented with a tabbed interface: the main counter, application settings, and credits. The main tab allows users to upload a folder of images, which are then sent to the backend for processing via a form submission. While images are being processed, a loading bar is displayed to indicate progress.

Once the backend returns the results, the JavaScript code dynamically updates the main results table with the detected dot counts for each image. Users can create and assign tags to images, filter previous results by tag, and export the main table to CSV. The interface also allows users to adjust dot counts manually, delete selected results, and manage tags in bulk.

The settings tab, loaded dynamically from a separate HTML file, lets users configure detection parameters and output options. Changes to settings are sent to the backend and saved in a YAML configuration file. The credits tab displays project credits and logos.

All user interactions—such as uploading images, adjusting settings, tagging, and exporting—are handled client-side with JavaScript, which communicates with the backend using fetch requests to various API endpoints. The frontend thus provides a responsive, interactive interface for managing image analysis and results.

## **API**

The FastAPI backend serves as the bridge between the frontend and the image analysis logic. When a user uploads images through the web interface, the `/detect_dots/` endpoint receives the files, saves them temporarily, and processes each image using the `BlackDotDetector` (with settings either from the config or predicted by a model). After processing, the backend reads the results from JSON files and returns them to the frontend, which updates the results table.

The backend also provides endpoints for configuration management: `/get_yaml` and `/update_yaml` allow the frontend to fetch and update the YAML settings file, while `/get_default_yaml` returns the default configuration. The `/get_image/{image_name}` endpoint serves processed images for viewing in the browser, and `/list_previous_images/` lists all processed images available in the output directory.

For result management, `/delete_results/` deletes selected images and their associated data. Tagging functionality is handled by `/add_tag/`, `/get_tags/`, and `/assign_tags/`, which allow the frontend to create, list, and assign tags to images. The `/adjust_dots/` endpoint lets users manually adjust dot counts for specific images, updating the corresponding JSON files.

Static files (such as processed images) are served from the output directory, and the root endpoint `/` serves the main HTML client. There is also a generic endpoint to serve additional HTML pages by name. The backend ensures that all file operations are safe, and that errors are reported clearly to the frontend, enabling a smooth and interactive user experience.

## **Discussion/conclusion:**

### **Discussion**

This project was developed as a learning experience, with the team starting with limited knowledge of image recognition and web development. Over the course of the project, we encountered challenges related to data storage and management. In hindsight, using a database for storing results and tags would have streamlined development and improved scalability. The current solution, which stores results in individual JSON files, works but can be cumbersome for large datasets. Similarly, the tagging system could be improved by allowing for more flexible filtering and export options.

Despite these limitations, the tool fulfills its primary goal of automating gold nanoparticle counting and provides a user-friendly interface for researchers. The modular design allows for future enhancements, and the open-source nature of the project invites contributions from the community.

### **Conclusion**

The Gold Nanoparticle Counter offers an efficient and reproducible solution for quantifying gold nanoparticles in microscopy images. Its combination of advanced image processing, an intuitive interface, and flexible data management makes it a valuable tool for researchers in cell biology and related fields. The application not only accelerates data analysis but also improves the reliability and reproducibility of results.

### **Possible Future Improvements**

While the current version meets the core requirements, several enhancements could further improve usability and functionality. For example, settings profiles could allow users to save and load different parameter sets for various experiments. The export functionality could also be upgraded to support future sorting by tags, include area and dots per area, and exporting to .xlsx format. Model training could also be added to the user interface, which would enable users to refine detection parameters the model predicts based on their own data, making the tool more adaptable. Additional features, such as reporting cell dimensions and dot distribution throughout the cell could provide deeper insights into results.

## Credits & References

### Project Team

Bram Koobs – b.l.koobs@st.hanze.nl

Michiel Meeuwisse – michiel.meeuwisse@gmail.com

### Supervision & Support

Dave Langers – Hanze University of Applied Sciences

Marcel Kempenaar – Hanze University of Applied Sciences

Sanne van der Niet – University of Amsterdam

Nichole van der Wel – University of Amsterdam

### Commissioned By

University of Amsterdam

### Special Thanks

Hanze University of Applied Sciences

University of Amsterdam

### Libraries & Tools Used

Numpy

FastAPI

OpenCV

NumPy

Matplotlib

EasyOCR

PyYAML

Pickle

### Contact

For questions, bug reports, or contributions, please contact the project authors at the emails above.

## References

- Franklin, A., Salgueiro, V. C., Layton, A. J., Sullivan, R., Mize, T., Vázquez-Iniesta, L., Benedict, S. T., Gurcha, S. S., Anso, I., Besra, G. S., Banzhaf, M., Lovering, A. L., Williams, S. J., Guerin, M. E., Scott, N. E., Prados-Rosales, R., Lowe, E. C., & Moynihan, P. J. (2024). The mycobacterial glycoside hydrolase LamH enables capsular arabinomannan release and stimulates growth. *Nature Communications*, 15(1). <https://doi.org/10.1038/s41467-024-50051-3>
- Matsunaga, I., Bhatt, A., Young, D. C., Cheng, T. Y., Eyles, S. J., Besra, G. S., Briken, V., Porcelli, S. A., Costello, C. E., Jacobs, W. R., & Moody, D. B. (2004). *Mycobacterium tuberculosis* pks12 produces a novel polyketide presented by CD1c to T cells. *Journal of Experimental Medicine*, 200(12), 1559–1569. <https://doi.org/10.1084/jem.20041429>
- Sani, M., Houben, E. N. G., Geurtsen, J., Pierson, J., De Punder, K., Van Zon, M., Wever, B., Piersma, S. R., Jiménez, C. R., Daffé, M., Appelmek, B. J., Bitter, W., Van Wel, N. Der, & Peters, P. J. (2010). Direct visualization by Cryo-EM of the mycobacterial capsular layer: A labile structure containing ESX-1-secreted proteins. *PLoS Pathogens*, 6(3). <https://doi.org/10.1371/journal.ppat.1000794>
- WHO. (2023). *Global Tuberculosis Report*. World Health Organization, 2–3.