Gold Nanoparticle Counter – User Manual

Overview

The Gold Nanoparticle Counter is a tool for detecting and counting gold nanoparticles labels in TEM images of tuberculosis cells. It provides an intuitive interface for uploading images, processing them with advanced detection algorithms, and managing results with tagging, adjustment, and export features.

Table of Contents

How to use:

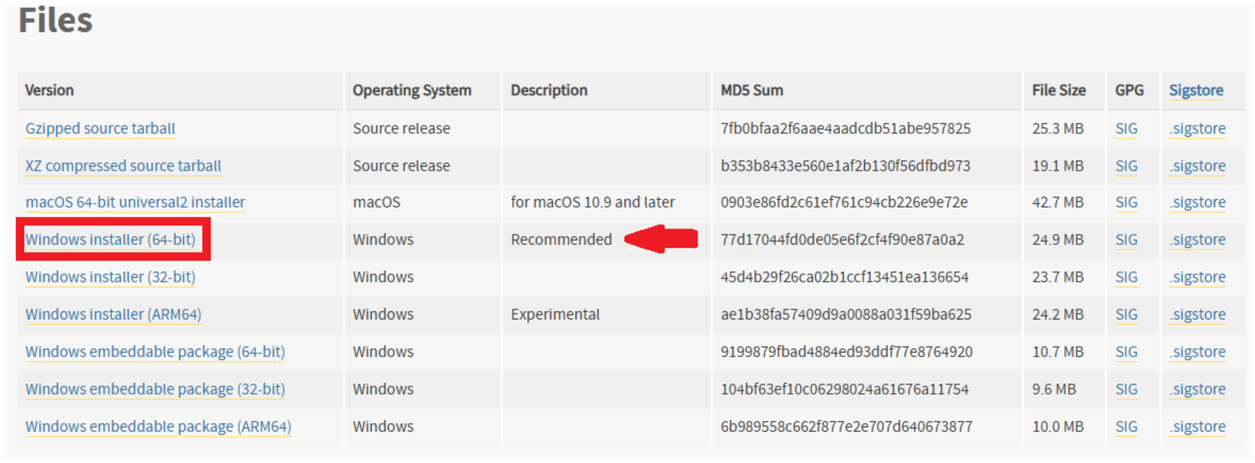
Getting Started

Step 1, install python:

The version of python that was used to create this application is 3.11.8. In order to install this version, 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
Gzipped source tarball	Source release		7fb0bfaa2f6aae4aadcdb51abe957825	25.3 MB	SIG	.sigstore
XZ compressed source tarball	Source release		b353b8433e560e1af2b130f56dfbd973	19.1 MB	SIG	.sigstore
macOS 64-bit universal2 installer	macOS	for macOS 10.9 and later	0903e86fd2c61ef761c94cb226e9e72e	42.7 MB	SIG	.sigstore
Windows installer (64-bit)	Windows	Recommended	77d17044fd0de05e6f2cf4f90e87a0a2	24.9 MB	SIG	.sigstore
Windows installer (32-bit)	Windows		45d4b29f26ca02b1ccf13451ea136654	23.7 MB	SIG	.sigstore
Windows installer (ARM64)	Windows	Experimental	ae1b38fa57409d9a0088a031f59ba625	24.2 MB	SIG	.sigstore
Windows embeddable package (64-bit)	Windows		9199879fbad4884ed93ddf77e8764920	10.7 MB	SIG	.sigstore
Windows embeddable package (32-bit)	Windows		104bf63ef10c06298024a61676a11754	9.6 MB	SIG	.sigstore
Windows embeddable package (ARM64)	Windows		6b989558c662f877e2e707d640673877	10.0 MB	SIG	.sigstore

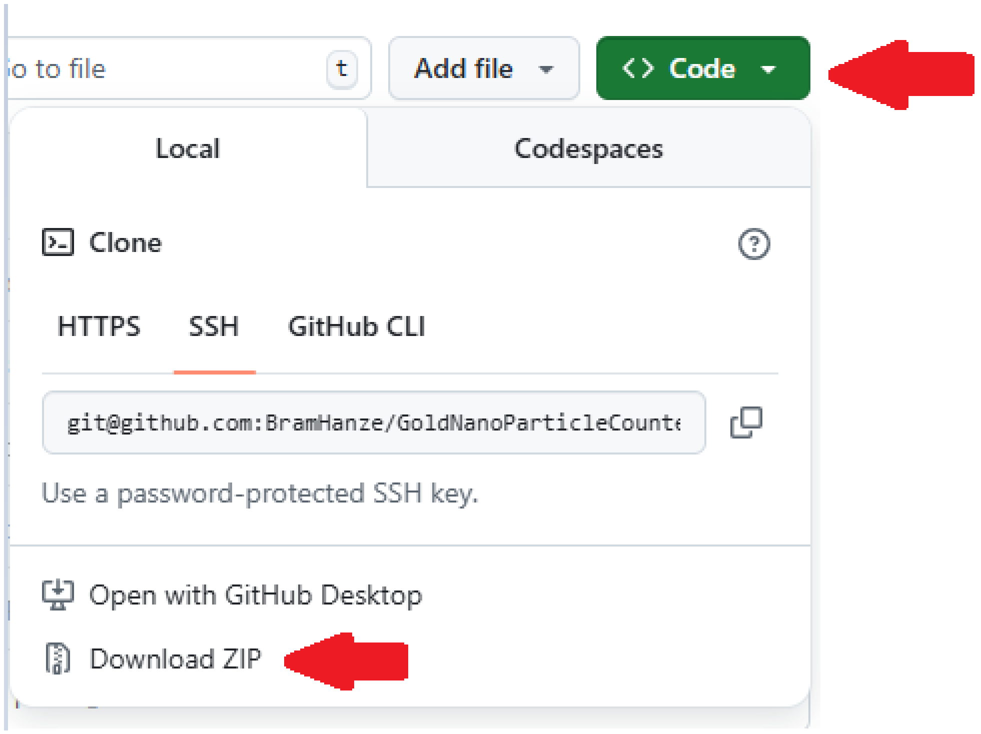
When you have downloaded the installer, you can open it up and follow all the steps until Python is installed.

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

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



```
git@github.com:BramHanze/GoldNanoParticleCount
```

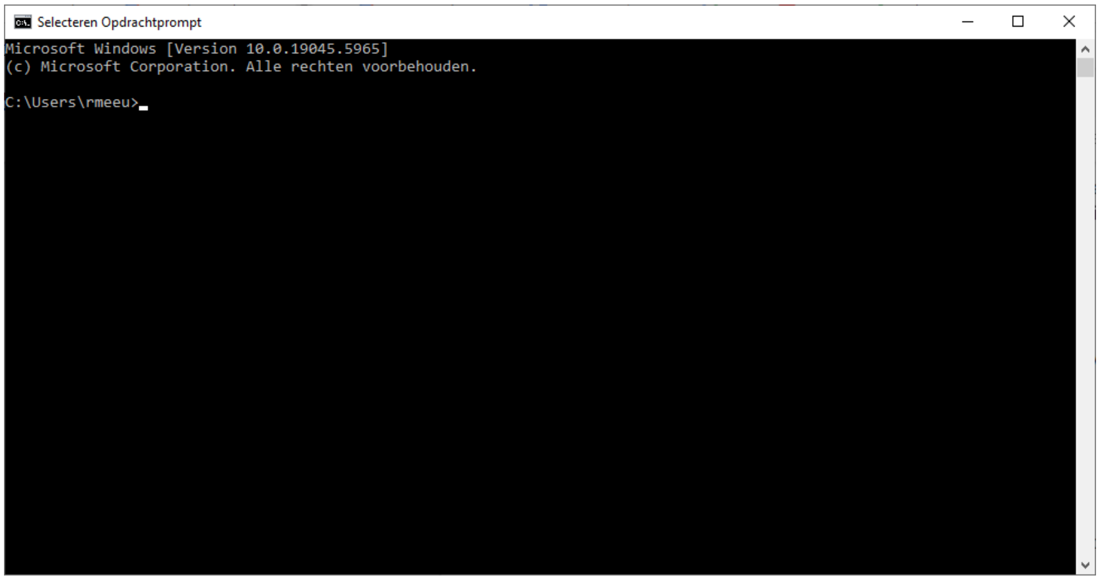
Use a password-protected SSH key.

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, install the needed dependencies:

When using Windows, you can open the Windows search bar by pressing the Windows key or by navigating to the bottom left of the screen. In this search bar, you type cmd and then you press enter. It should open a window that looks something like this



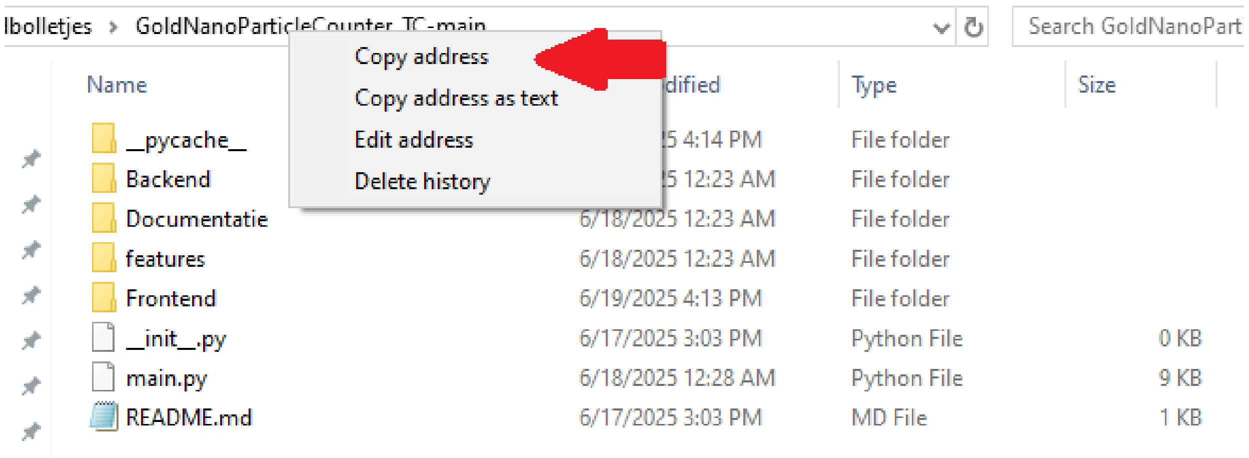
```
Microsoft Windows [Version 10.0.19045.5965]
(c) Microsoft Corporation. Alle rechten voorbehouden.

C:\Users\rmeeu>
```

In this window you type the following commands one by one in order to install all the needed dependencies.

Pip3 install fastapi[all]
Pip3 install opencv-python
Pip3 install matplotlib
Pip3 install numpy
Pip3 install scikit-learn
Pip3 install easyocr

Step 4, Run the application:

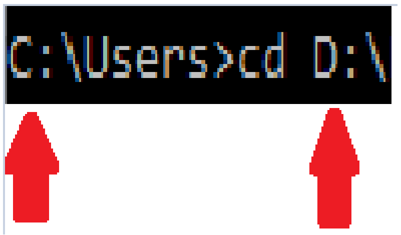
In order to run the application, you will have to navigate to the correct folder. In order to do this you first locate the GoldNanoParticleCounter_TC-main folder in your filebrowser. When you are in the correct folder you right click the folder in the navigation bar at the top and click the option copy address.



Name	Date modified	Type	Size
__pycache__	5 4:14 PM	File folder	
Backend	5 12:23 AM	File folder	
Documentatie	6/18/2025 12:23 AM	File folder	
features	6/18/2025 12:23 AM	File folder	
Frontend	6/19/2025 4:13 PM	File folder	
__init__.py	6/17/2025 3:03 PM	Python File	0 KB
main.py	6/18/2025 12:28 AM	Python File	9 KB
README.md	6/17/2025 3:03 PM	MD File	1 KB

When you have copied the address, you go back to your cmd promt and type ”cd” (change directory) then you paste the address by pressing Crtl+V and then press enter.

If nothing happens it could be that you are on the wrong drive if this is the case the letters followed by a ”:” will be different like this:



```
C:\Users>cd D:\
```

If this is the case you type “cd /d” (in this case its /d because the second drive is the D drive, change this letter to whatever your drive is called) and then paste the adress with Crtl+V. And press enter

You should now be in the correct folder. You can now easily run the application by typing the following command:

Fastapi dev Frontend/main.py

If everything went well your aplication should now start up. In order to access your application you can click the link in the terminal or go to this url:

http://127.0.0.1:8000

Uploading and Processing Images

Upload Folder

Click the "Upload Folder" button and select a folder containing your microscopy images.

Supported image input format: .tif

Supported image output formats: .jpg, .png, .jpeg

Process Images

Click "Process Images" to start analysis.

A loading bar will indicate that the backend has received and is processing the image(s).

The application will skip images already processed before if "Only Run New Images" is enabled in settings.

Viewing and Managing Results

Main Table

Displays results/info for the currently selected images:

- Image Name
- Normal Dots: Single detected nanoparticles.
- Cluster Dots: Estimated count in clusters.
- Total Dots: Sum of normal and cluster dots.
- Adjust Dots: Manual adjustment field.
- Tags: Assigned tags.
- Area nm^2: Detected cell area.
- Dots/nm^2: Density calculation.
- View Image: Opens the processed image.

Select All: Select or unselect all rows for bulk actions.

Clear Table: Removes all rows from the main table (does not delete files).

Previous Runs

- Shows all previously processed images.
- Filter by tag using the dropdown.
- Add to Main Table: Move selected results to the main table for further analysis.
- Delete Selected: Permanently deletes selected results (image and JSON).

Tagging System

Create Tag:

- Enter a tag name and click "Create Tag".

- Tags are managed globally and can be assigned to any image.

Assign Tags:

- Select images in the main table.
- Select one or more tags from the "Select Tag(s)" dropdown.
- Click "Add Tags" to assign.

Clear Tags:

- Select images and click "Clear Tags" to remove all tags from those images.

Filter by tag:

- In the Previous Runs panel the user can filter the images by tag.

Adjusting Dot Counts

- In the main table, use the "Adjust Dots" field to manually correct the total dot count for any image.
- The adjusted value is automatically saved, and changing this value causes the "Total Dots" and "Dots/nm^2" columns to update its value.

Exporting Data

Export to CSV:

- Click the "Export to CSV" button in the main table to download all current results as a CSV file.
- The CSV includes: Image, Normal Dots, Cluster Dots, Total Dots, Adjust Dots, Tags.

Settings

Access the Settings tab to customize detection and output parameters.

General Settings:

- Predict best settings, minimum dot area, dot blur, etc.

Advanced Settings:

- Circularity thresholds, cluster detection, dynamic thresholds, etc.
- Toggle "Show Advanced Settings" to reveal more options.

Output Settings:

- Output directory, image format and color settings for the found dots.

Cell Detection & Dot Detection:

- Adjust parameters for cell and dot identification.

Reset:

- All inputs besides checkboxes have a "Reset" button to revert individual fields to default values.

Submit:

- Click "Submit" to save changes.

Credits

- Project by Bram Koobs and Michiel Meeuwisse
- Hanze University of Applied Sciences & University of Amsterdam
- See the Credits tab in the app for full acknowledgments and contact details.

Troubleshooting

Images not processing:

- Ensure images have not been processed before and

Settings not saving:

- Click checkboxes to expand advanced settings before submitting.

Tag or adjustment not updating:

- Refresh the page or restart the backend/server

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.

Backend/

- **blackdotdetector.py**: Core detection logic.
- **scale_finder.py**: Automatic scale detection.
- **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. Throughout 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 made while we had very little knowledge of concepts like image recognition and JavaScript. So, making this project was a learning experience for us. We had to learn while we were making the application. Because of this there are several parts of our application that we would have changed if we started with the same knowledge we have right now. One of the main things we would change is the way we store our data. Right now, we have one folder that contains all the processed images and json files that contain all the information for the processed images. However, this has caused some slight issues during development because the information is stored in a lot of different files. If we had created a small database to store the information, this would have prevented problems later on in development. Even though all problems have eventually been solved with our current solution. It could have been done a lot easier and more efficiently if we had spent some time in the beginning to make a simple database. Another thing we would change if we started from the beginning again is the way our tags are stored. Right now, the tags an image has is just a list of tags. This doesn’t cause any problems within the application, but it makes it harder to find images by their tags when the results are exported to a csv because the tags collum just contains whatever tags the image has. You could have created separate columns for each tag that is in the csv with a true or false in the column indicating if the tag is attached to the image. This way you can sort through your images based upon the selected tag. However, we are still happy with the product we have produced. Even though it is not perfect it performs well for the function it was intended for. There are certain things that we would have liked to implement if we had more time. These are discussed later on.

Conclusion:

The Gold Nanoparticle Counter provides an efficient and user-friendly solution for the automated detection and counting of gold nanoparticles in microscopy images. It utilizes advanced image processing techniques with an intuitive interface. The tool can be used for the processing itself, but also for sorting the data. This tool makes the counting replicable, enhancing reproducibility. The flexible tagging system, and export features make the tool useful for more than just the detecting and counting of gold nanoparticles.

Possible Future Improvements:

- Settings Profiles:

Enable users to save and load different settings profiles based on the experiment type or imaging conditions, possibly improving the workflow.

- Adjustments/Improvements to the Export Functionality:

Enhance the export options to allow sorting by tags in excel, giving the user the choice between .csv and .xlsx, and having an option to change the download directory for the export, making it easier to access the results with external applications.

- Model Training Through the User Interface:

Allow the user to train the model on a specific image (and handpicked settings) via the web interface, allowing the model to improve over time, thus making the model (and this tool) more future proof.

- More information about the cell and dots:

Possible features that could be included are the height and width of the cell, and the distribution of the dots along the length of the cell.

- Grouping of adjoining dots, and clusters

Multiple closely grouped dots likely represent binding to a single protein, indicating the presence of most likely only one protein at that location.

Credits & References

Project Team

- Bram Koobs

Email: b.l.koobs@st.hanze.nl

- Michiel Meeuwisse

Email: 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

- FastAPI – Web framework for the backend
- OpenCV – Image processing
- NumPy – Numerical operations
- Matplotlib – Visualization
- EasyOCR – Optical character recognition
- PyYAML – YAML settings storing
- Pickle – Storing ML model

Contact

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