

AutoImageSeg V1.0.0 User Guide

1 Introduction

AutoImageSeg is a zero-code, open-source image segmentation software designed to make advanced image segmentation techniques accessible to users with no programming background. By integrating nine mainstream PyTorch models, it provides a comprehensive workflow from training to inference, evaluation, and annotation, all through an intuitive graphical interface.

2 Configuration File Settings

The configuration file of the software mainly includes several parts: **common**, **training**, **validation**, **labelme_conversion**, and **restriction**.

In the **common** section, it mainly contains **seed** and **image size**. By adjusting the **seed**, the random seed for training the model can be set, thereby enhancing the reproducibility of the model. Since the sizes of the training images are inconsistent, the software will uniformly resize the input images to a specific square resolution during training and inference. The default **image size** is set to 512, which should be smaller than the shortest side length of the images used for the task.

In the **training** section:

- The **data_augmentation** part includes settings such as **clahe**, **flip**, **rotate**, **random blur**, **random brightness**, and **random contrast**.
- The **dataloader** part allows settings for **num workers**, **batch size**, and **shuffle**. The default **num workers** is set to 0, and to avoid errors, users need to modify it in the code. During training, the default **shuffle** is set to **true**, which helps to better alleviate overfitting.
- Additionally, the training process can set the **evemetric**, with options being **dice** or **iou**. The corresponding formulas are as follows, which are the most widely used evaluation metrics in image segmentation tasks:

$$\text{Dice} = \frac{2 \times \text{TP}}{2 \times \text{TP} + \text{FP} + \text{FN}} \quad (1)$$

$$\text{IoU} = \frac{\text{TP}}{\text{TP} + \text{FP} + \text{FN}} \quad (2)$$

- The **earlystop** training setting defaults to 0. If greater than 0, during training, if there is no increase in the corresponding **evemetric** for **earlystop** epochs, it will automatically stop without having to run through all the epochs set by the software. Additionally, **lr** represents the learning rate, which defaults to 1×10^{-4} in this software.

In the **validation** section:

- For **data_augmentation**, only the **CLAHE** setting is retained. If **CLAHE** is used during training, it is recommended to maintain consistency in the validation process.
- The **num workers** in the **dataloader** is recommended to be kept at 0, i.e., using only one worker. If modification is needed, it should be done in the code.
- Additionally, the **batch size** for the validation dataset can also be set.

In the **labelme_conversion** section, it mainly sets the configuration for converting the mask to LabelMe JSON after validation.

- **threshold_image** is the maximum number of images that the software can process at once when converting to JSON. This is to prevent the software from freezing, as in some models and tasks, if the model training results are not satisfactory, the inferred image masks may contain many pixel - level scatter errors, which can lead to incorrect JSON structures and significant computational load. Therefore, it is recommended to start with a small number of images for validation, especially when converting masks to color images on the post - processing page, to actually check the quality of image segmentation inference.
- In the **min_area** setting, small structures corresponding to pixel points will be automatically ignored. The default setting for converting LabelMe JSON is 10 pixel points.

- **dp_tolerance** represents the Douglas-Peucker tolerance in scikit-image's `measure.approximate_polygon`, which is an absolute pixel unit and defaults to 1.5px. Increasing **dp_tolerance** can reduce the number of vertices, thereby reducing the file size of LabelMe JSON and the subsequent manual editing interaction cost. Together with **min_area** to filter out small polygons, these two settings form the "lightweight post-processing" phase of AI-assisted segmentation: they can reduce the total number of vertices in a single image and significantly reduce the workload of annotators adjusting redundant nodes. Therefore, it is recommended to dynamically increase **dp_tolerance** to 3–5 px and set **min_area** to 10–20 px in a highly automated production line to balance geometric accuracy and annotation efficiency.

Additionally, the **restriction** section in the configuration file is the global software image limitation.

- **image_max_length** is the maximum allowed length of any side of the images during training and validation, which defaults to 3000.
- **train_count**, **test_count**, **infer_count**, and **post_count** are the maximum number of images allowed for different tasks of the software. When running in software form, it is recommended that these **count** values do not exceed 3000.

3 Loss Functions Selection

The selection of loss function is crucial for the performance of the model in image segmentation tasks. Different loss functions can be optimized for different problems and data characteristics. In this software, we provide four commonly used loss functions for users to choose from, which have been widely used in image segmentation tasks. The detailed descriptions and mathematical formulas of these loss functions are as follows:

1. CrossEntropyLoss

Cross-entropy loss is one of the most commonly used classification loss functions, which measures the difference between the model's predicted probability distribution and the true label's probability distribution. In the task of image segmentation, cross-entropy loss can effectively optimize pixel-level classification.

$$\mathcal{L}_{CE} = -\sum_i y_i \log(p_i) \quad (3)$$

where y_i is the true label, and p_i is the model's predicted probability.

2. CE_DiceLoss

The combination of cross-entropy loss and Dice loss, \mathcal{L}_{CE_Dice} , aims to optimize both classification accuracy and segmentation boundary precision simultaneously. Dice loss is particularly suitable for segmentation tasks because it directly optimizes the overlap of the segmented regions.

$$\mathcal{L}_{CE_Dice} = \mathcal{L}_{CE} - \lambda \cdot \mathcal{L}_{Dice} \quad (4)$$

where \mathcal{L}_{CE} is the cross-entropy loss, \mathcal{L}_{Dice} is the Dice loss, and λ is a balancing parameter.

The formula for Dice loss is:

$$\mathcal{L}_{Dice} = 1 - \frac{2\sum_i p_i y_i + \epsilon}{\sum_i p_i + \sum_i y_i + \epsilon} \quad (5)$$

where p_i is the model's predicted probability, y_i is the true label, and ϵ is a smoothing term to avoid division by zero.

3. Focal Loss

Focal Loss is an improved version of cross-entropy loss, designed to address the problem of class imbalance. It reduces the loss weight of easily classified samples, focusing the model's attention on hard-to-classify samples.

$$\mathcal{L}_{Focal} = -\alpha (1 - p_t)^\gamma y_i \log(p_i) \quad (6)$$

where $p_t = p_i$ if $y_i = 1$, otherwise $p_t = 1 - p_i$, α is the class weight, and γ is the focusing parameter.

4. Lovasz Softmax

Lovasz Softmax is an improved version based on Lovasz Hinge loss, suitable for multi-class segmentation tasks. It optimizes the intersection over union (IoU) of segmentation directly by optimizing the Lovasz extension.

$$\mathcal{L}_{\text{Lovasz}} = \sum_i \text{Lovasz}(\hat{y}_i, y_i) \quad (7)$$

where \hat{y}_i is the model's predicted probability, and y_i is the true label. The specific form of Lovasz Softmax loss is relatively complex, involving the computation of the Lovasz extension.

These loss functions each have their own advantages and are suitable for different image segmentation tasks. Users can choose the appropriate loss function based on specific requirements and data characteristics to optimize the performance of the model.

4 Basic Train Guide

Firstly, take the Plant Segmentation Dataset as an example.

We have the following original images:



Figure 1. Original images, groundtruth masks, and editable LabelMe annotations from three public datasets.

We also have the following standard Groundtruth images:



Figure 2. Groundtruth images.

For these Groundtruth images, keep the background with a pixel grayscale value of 0. Set the segmentation area to a pixel grayscale value of 1. This will create a Groundtruth image that is indistinguishable to the naked eye but actually has grayscale values in the range [0,1].

Note to keep the names of the original images and masks paired. Allocate the training set and test set in an 8:2 ratio to obtain <https://github.com/AI-thpremed/AutoImageSeg/blob/main/datasets/plant.zip>.

Then we can create https://github.com/AI-thpremed/AutoImageSeg/blob/main/images/label_mapping_sample.json, which is this label JSON file. The JSON format is as follows:

```
{
  "TARGET": 1,
  "_background_": 0
}
```

If there are more target areas, you can start from 1 and arrange them in sequence.

After that, we can proceed with mask-based training:

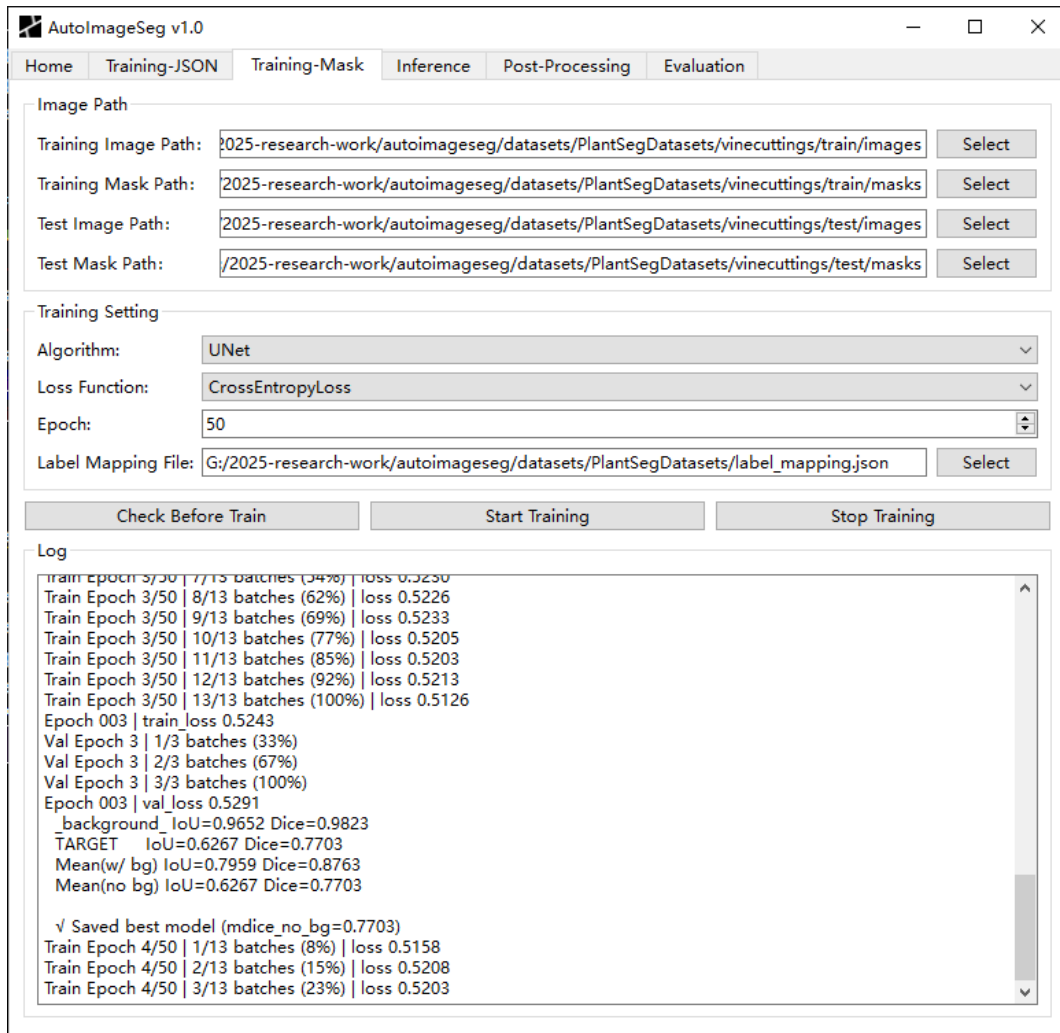


Figure 3. Training process.

All training results, including the best checkpoints based on the current model, the backup of the label mapping file, the training log, and the training config file, will be saved in the results folder.





 best.pth	2025/10/4 12:58	PTH 文件	121,354 KB
 label_mapping.json	2025/10/4 12:57	JSON File	1 KB
 training.log	2025/10/4 12:58	文本文档	23 KB
 training_config.json	2025/10/4 12:57	JSON File	1 KB

Figure 4. The training results are saved in the results folder.

Based on this saved training folder, we can proceed with subsequent inference, post-processing, and evaluation tasks. Each of these workflows will generate an independent folder saved within the results folder.

5 From Training Results to Labelme Editable Files

In the previous section, we established the results address for image segmentation by selecting the model and constructing the train and test sets. Next, we demonstrate how to use the segmentation model to generate Labelme-editable segmentation files for new images.

The workflow is illustrated in the following figure.

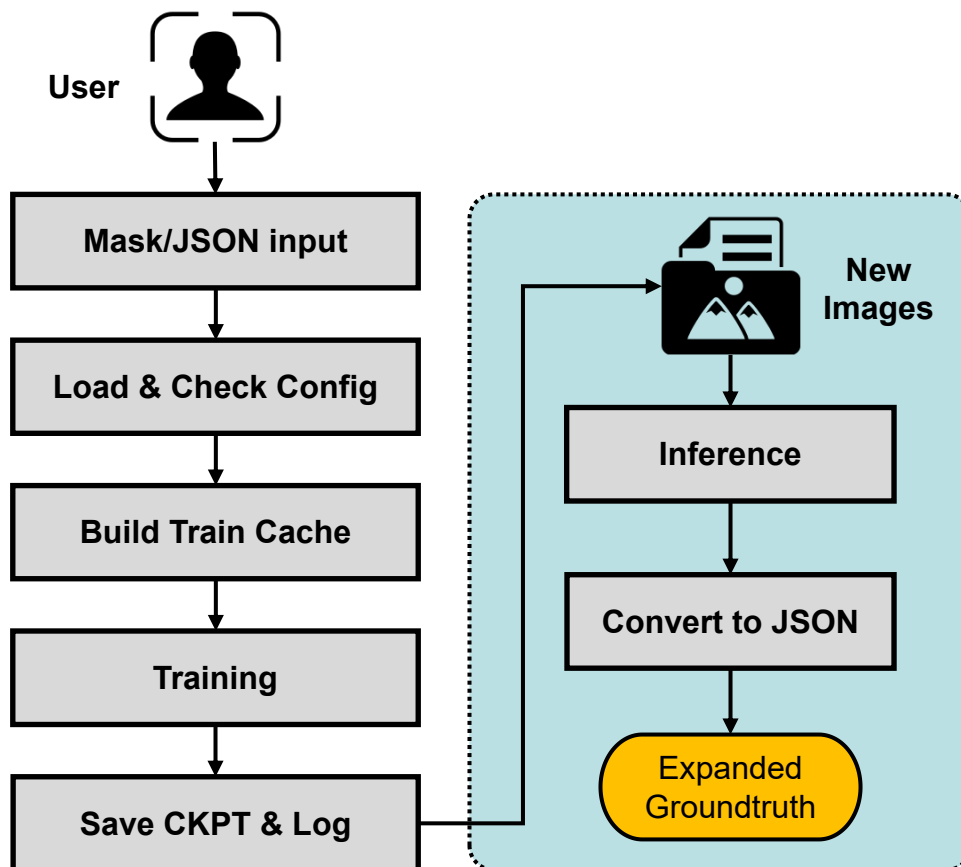


Figure 5. Pipeline for expanding segmentation groundtruth via model training and inference.

First, open the Inference page of the software. Then, in the Model Result section, select the address of the training results folder located in the results folder of our software. Additionally, input the address of a new set of images. We will read the training results and perform inference on the new images.

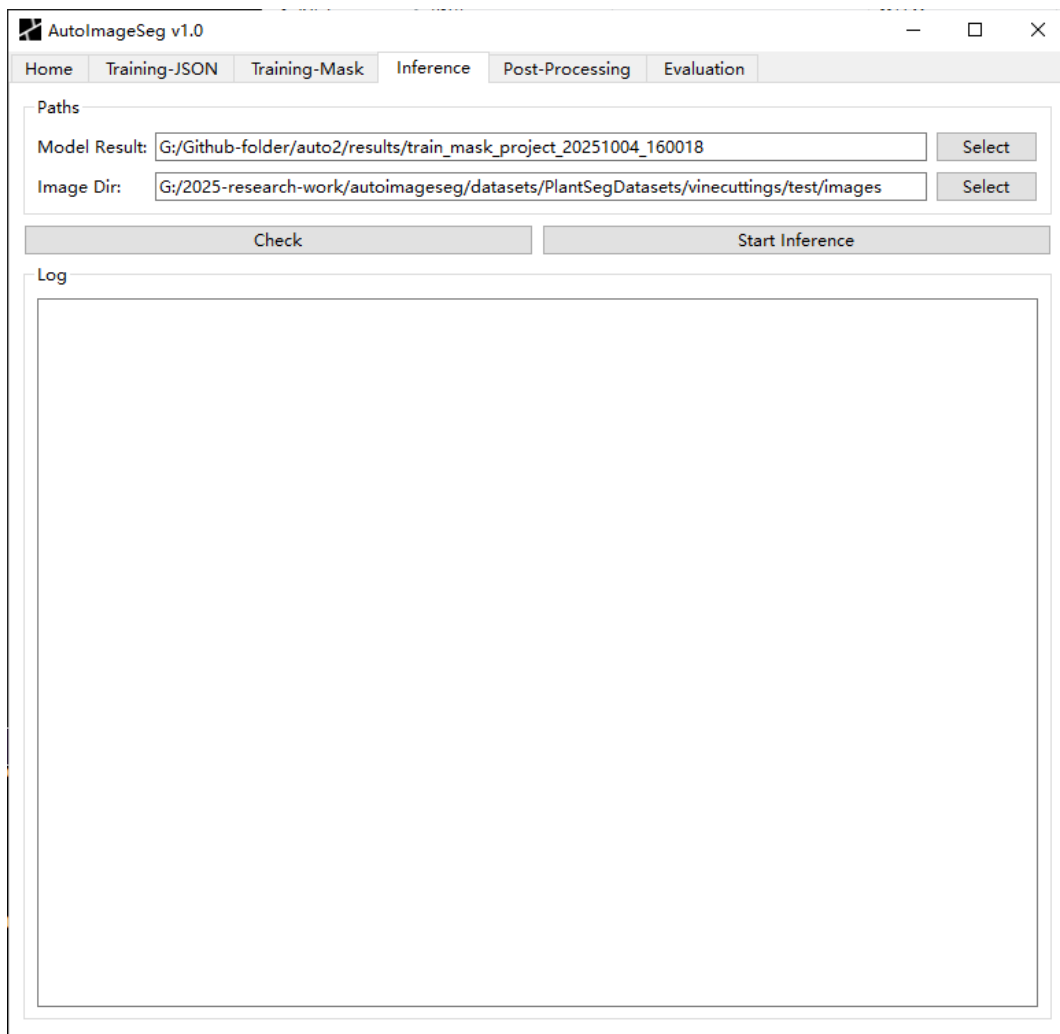


Figure 6. The training results are saved in the results folder.

Click the Check button. Once the conditions are met, click the Start Inference button to perform inference on the new images. This inference process will generate a file starting with infer in the software's results folder. The inference_results folder within it contains the mask images for the new images.

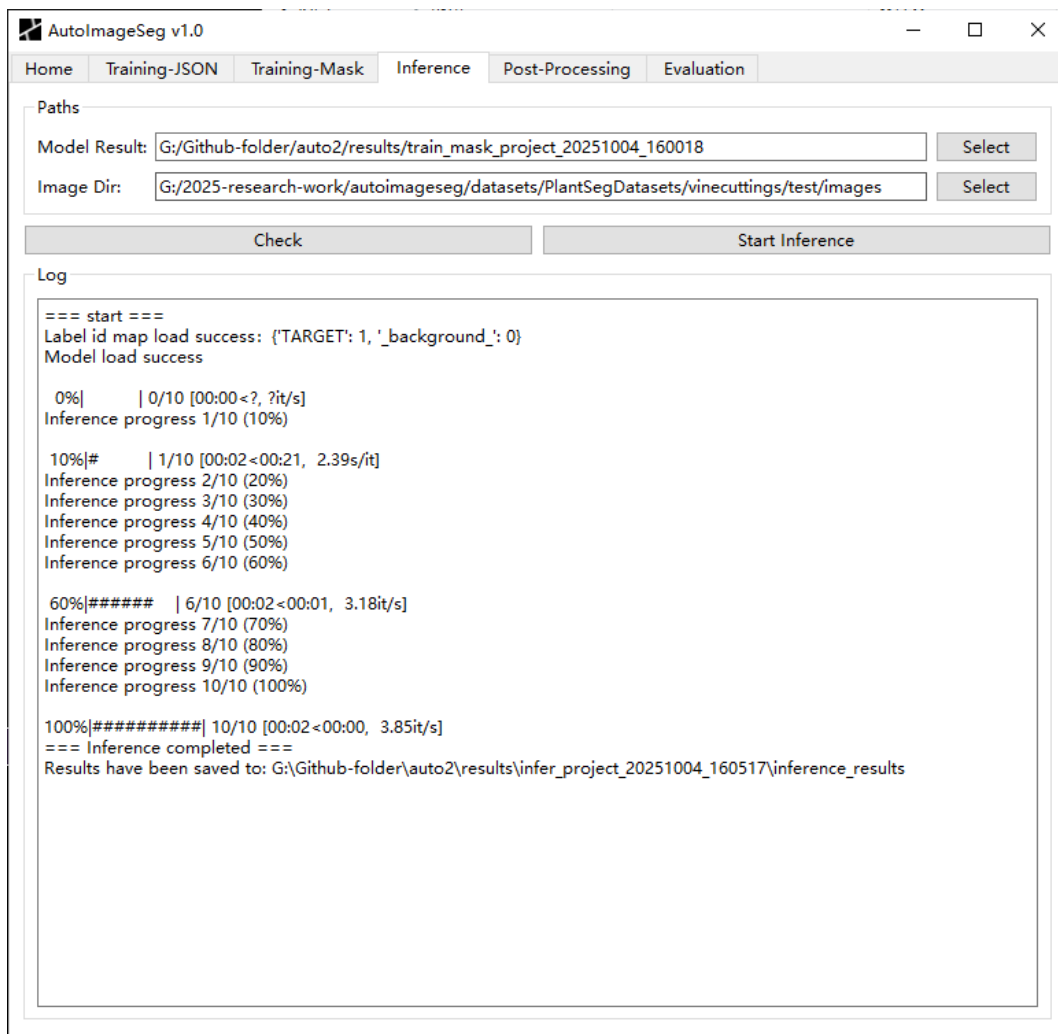


Figure 7. Inference results.

Open the Post-Processing page, then input the address of the Inference_results folder and the corresponding label_mapping file. Click Transfer Masks to Color Images to obtain the visualization results.

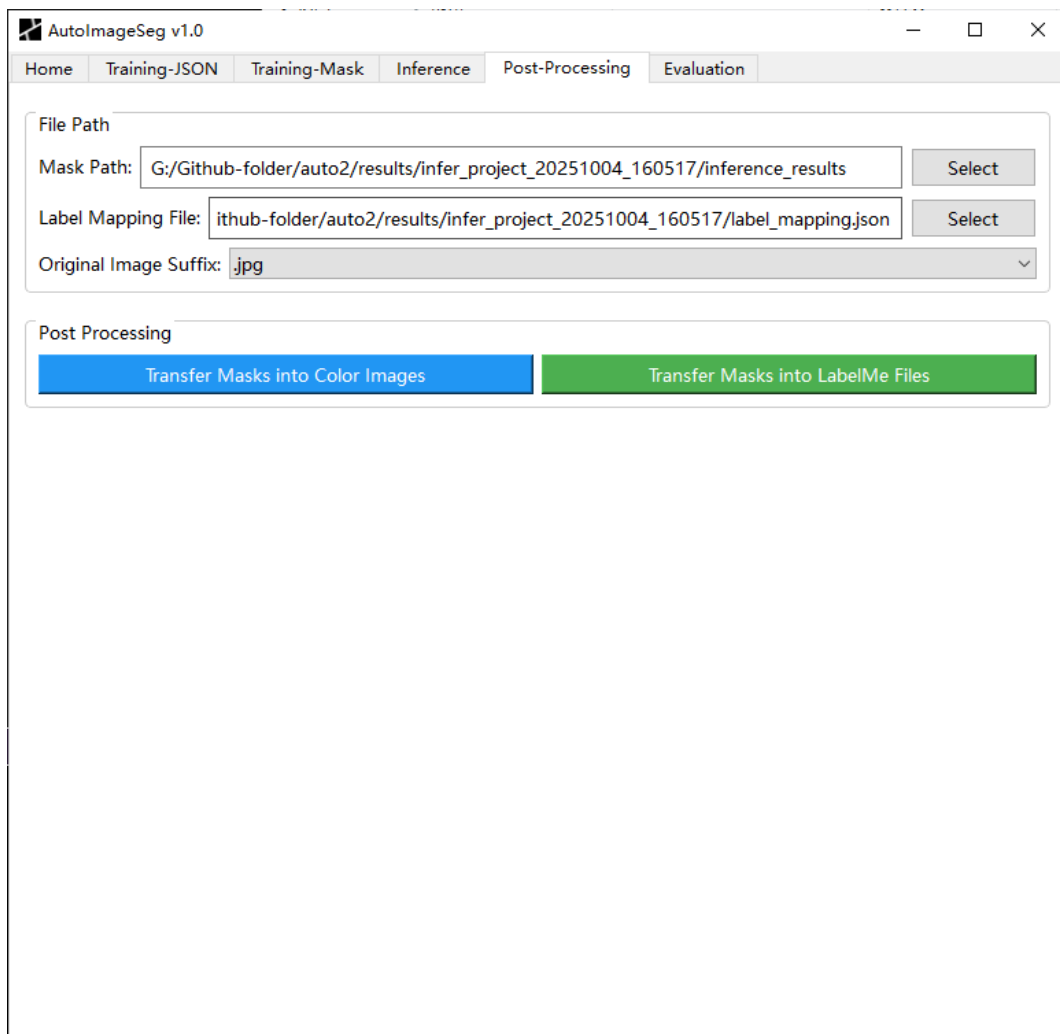


Figure 8. The Post-Processing page.

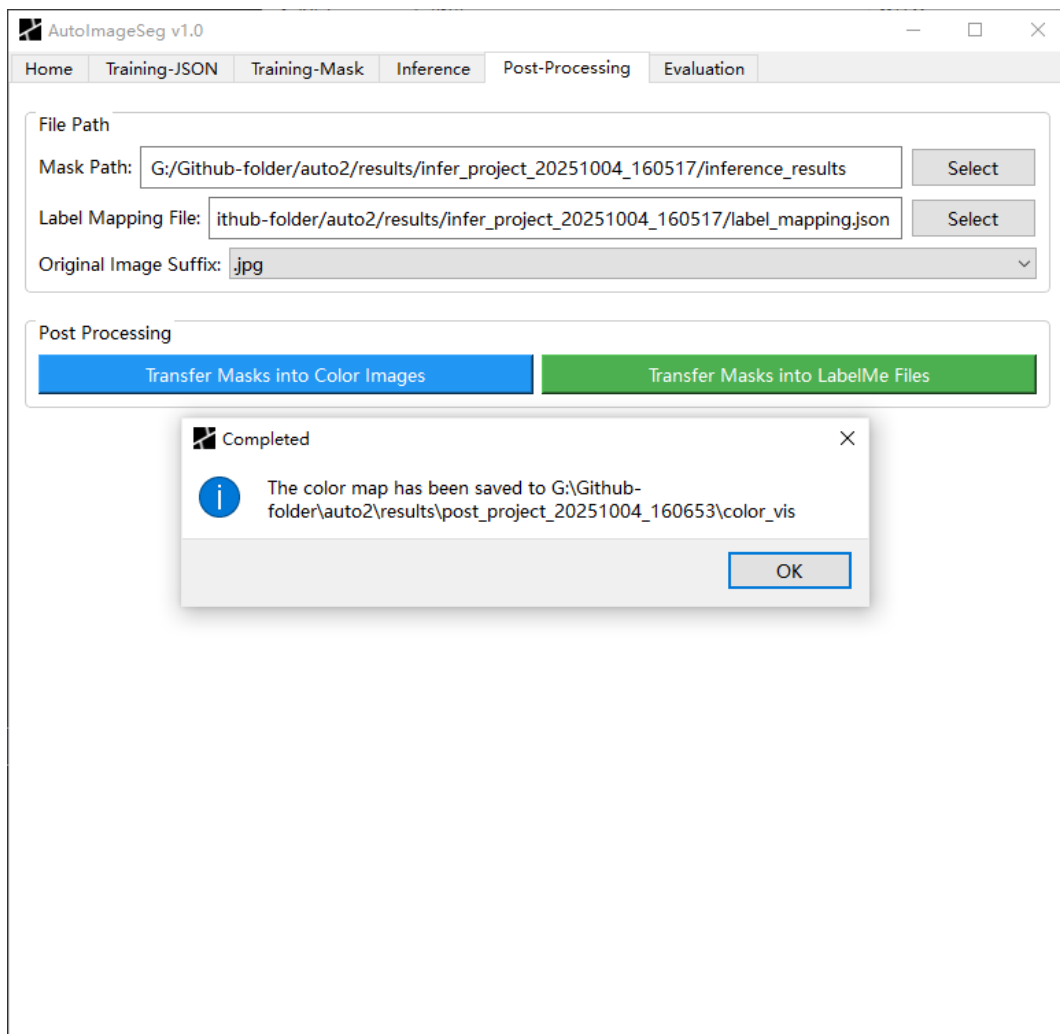


Figure 9. Convert masks into color images.

The visualization results are also saved in a folder within the results folder that starts with post. Inside this folder, there will be a folder named color_vis. The rule for converting images to color is configured according to the software's config-color file, which can be customized.



Figure 10. Visualization results of the color images.

Click the software's 'Transfer Masks into LabelMe Files' option to convert the Mask images into editable JSON files. Note that you need to set a suffix that pairs with the original images. We set it to .jpg files here.

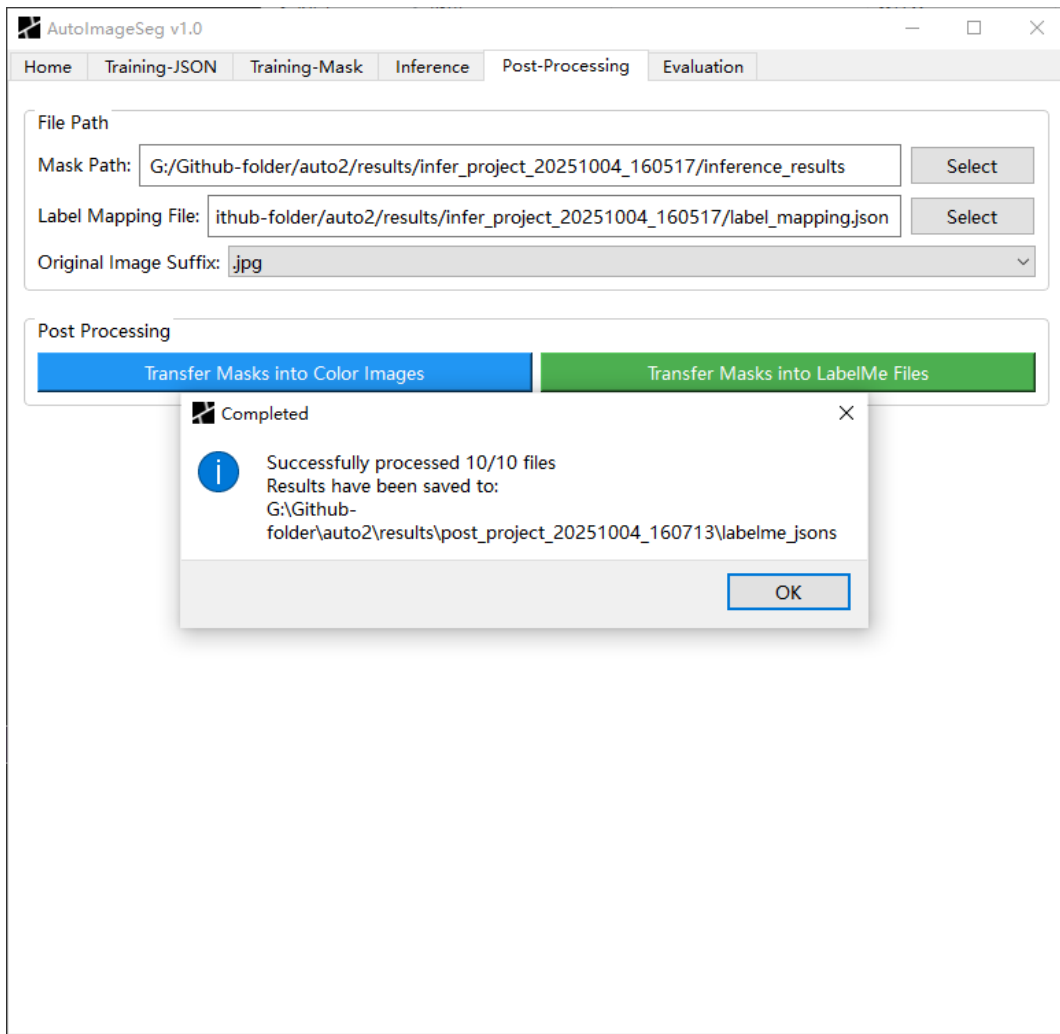


Figure 11. Successfully processed new images into JSON files.

After obtaining the corresponding LabelMe JSON files, place the original images and the LabelMe JSON files in the same folder.

DSC00235.json	2025/10/4 16:07	JSON File	77 KB
DSC00236.json	2025/10/4 16:07	JSON File	74 KB
DSC00238.json	2025/10/4 16:07	JSON File	80 KB
DSC00239.json	2025/10/4 16:07	JSON File	64 KB
DSC00241.json	2025/10/4 16:07	JSON File	91 KB
DSC00242.json	2025/10/4 16:07	JSON File	59 KB
DSC00243.json	2025/10/4 16:07	JSON File	82 KB
DSC00247.json	2025/10/4 16:07	JSON File	63 KB
DSC00248.json	2025/10/4 16:07	JSON File	107 KB
DSC00252.json	2025/10/4 16:07	JSON File	71 KB

Figure 12. The editable JSON files are saved in the results folder.

As shown in the figure below, we have achieved AI-based segmentation and constructed editable segmentation files.

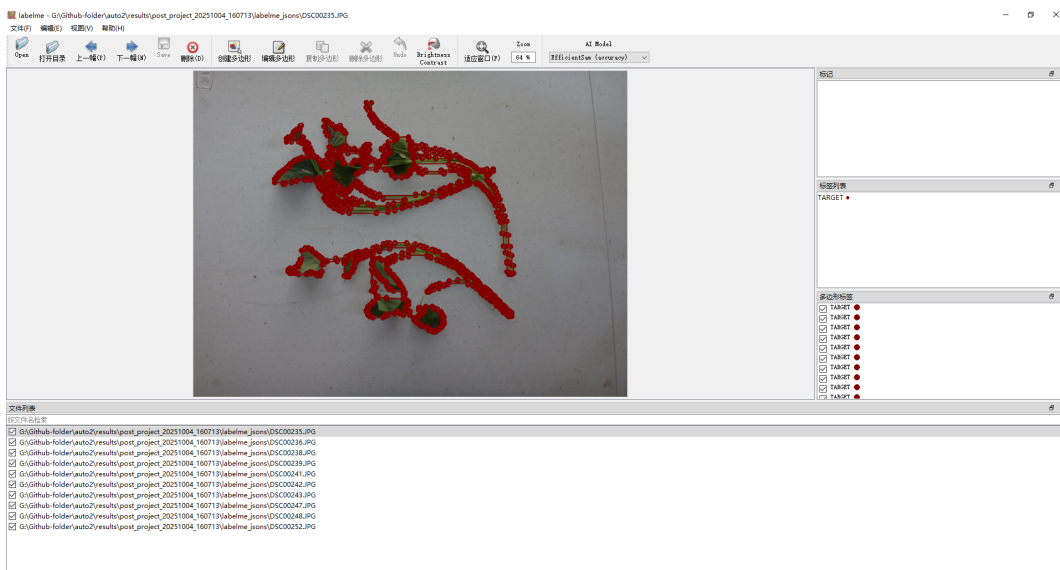


Figure 13. Opening our segmentation results with LabelMe software.

During this process, we have four work results in the results folder: the train-mask folder, the Inference folder, and two Post-Processing folders.





	infer_project_20251004_160517	2025/10/4 16:05	文件夹
	post_project_20251004_160653	2025/10/4 16:06	文件夹
	post_project_20251004_160713	2025/10/4 16:07	文件夹
	train_mask_project_20251004_160018	2025/10/4 16:01	文件夹

Figure 14. Results folder.

6 Conclusion

All code is open-source and licensed under the Apache 2.0 license. If you have any questions or need further assistance, feel free to contact me at weihaomeva@163.com. For those who are interested in custom development or support, I am available for paid consultations.

The code repository is available at: <https://github.com/AI-thpremed/AutoImageSeg>