UNIVERSITÄT DES SAARLANDES

dfki ai

Integrating ImageJ/Fiji Image Processing in Chaldene Visual Programming System

Ziwei He

Master Thesis Seminar

March 22, 2024,

Saarbrücken, Germany

Background - ImageJ/Fiji

ImageJ/Fiji

- **ImageJ**, is a Java-based software widely used for microscopy image processing in material sciences and bioinformatic field.

- **Fiji** is a "batteries-included" distribution of ImageJ, which is proposed as a platform for productive collaboration between computer science and other diverse research communities. [1]

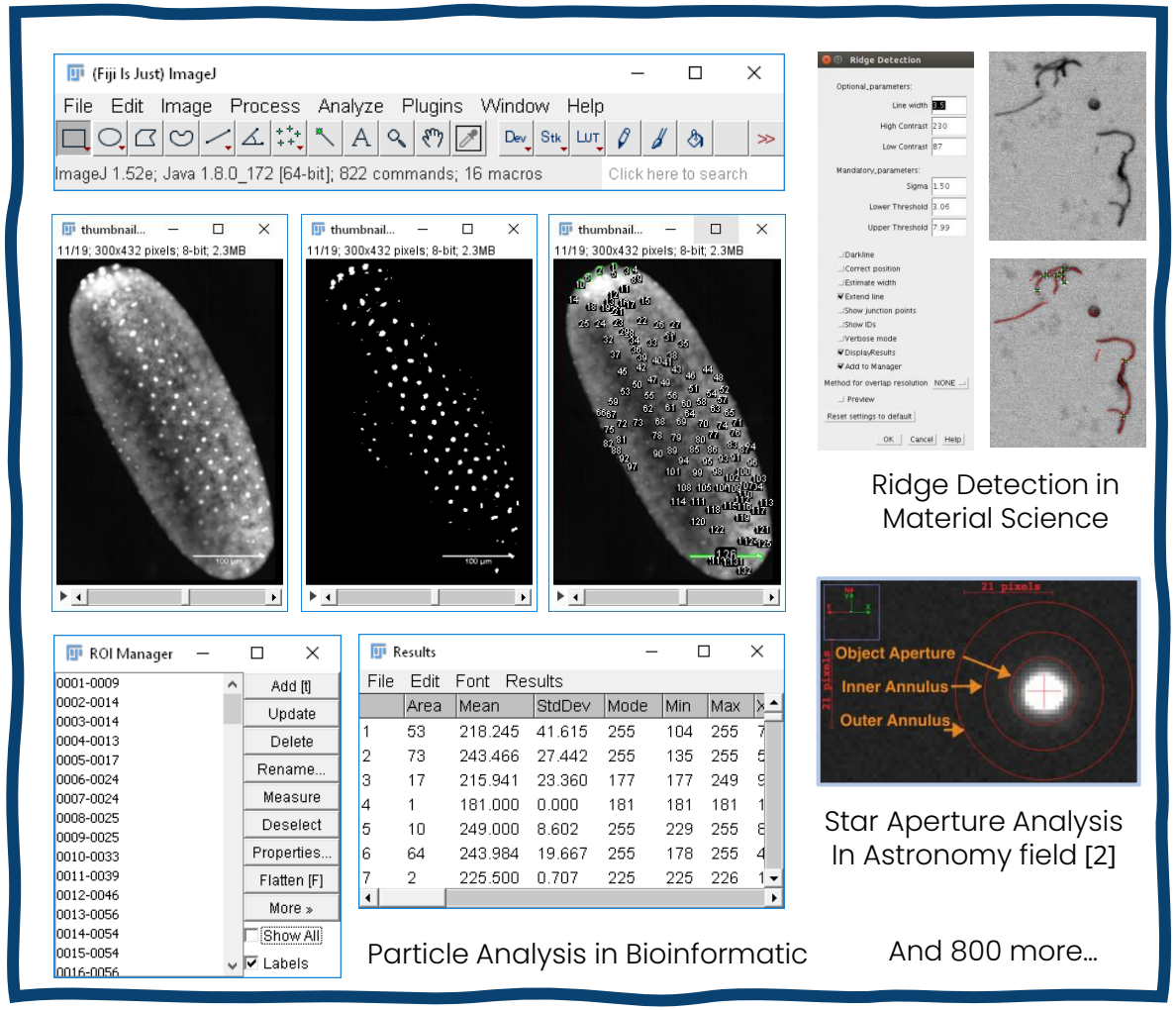


Figure 1 : Diverse Applications of ImageJ/Fiji Plugins Across Research Communities [3]

Background - Chaldene

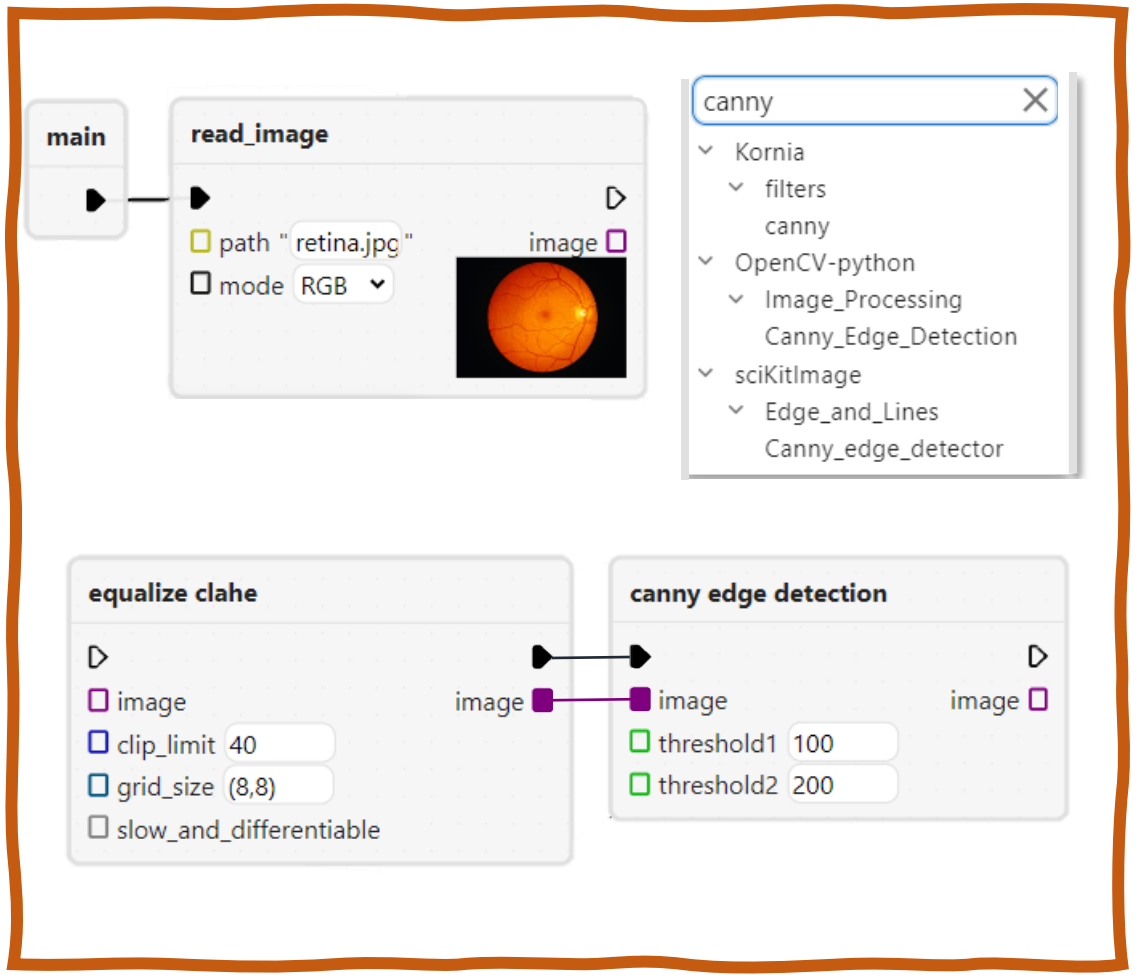


Figure 2 : Visual Nodes, Search Menu, Workflow in Chaldene

Chaldene
Visual Programming System

- **Chaldene** [4] is a visual programming extension to JupyterLab that executed based on Python kernel.
- **Chaldene** intends to provide scientists who have limited programming experience with a visual workflow for data processing, making complex data analysis more accessible and intuitive.

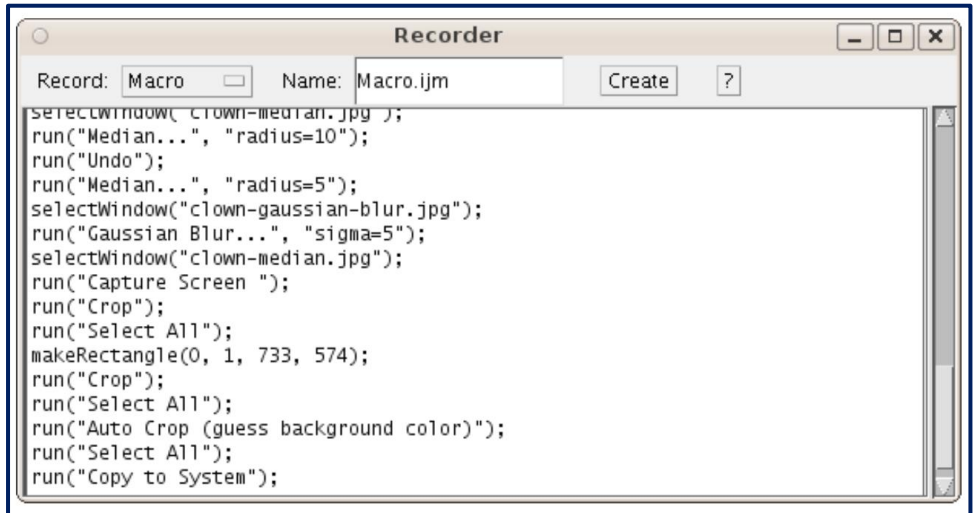
Motivation - Comparison

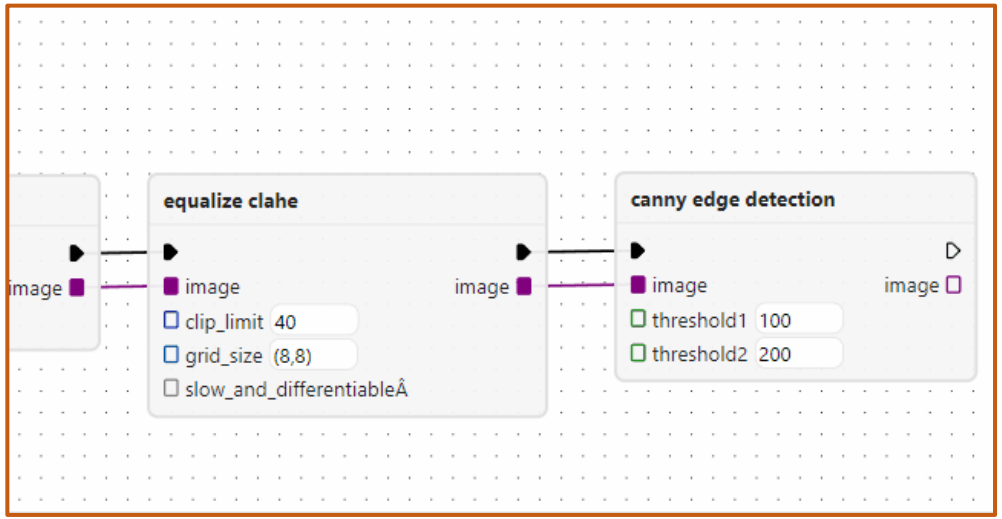
ImageJ/Fiji

- Java-based software
- ✓ Widely used
- × Confined to Java Eco
- × Hard to share and extend
  - × Macro recorder[5]

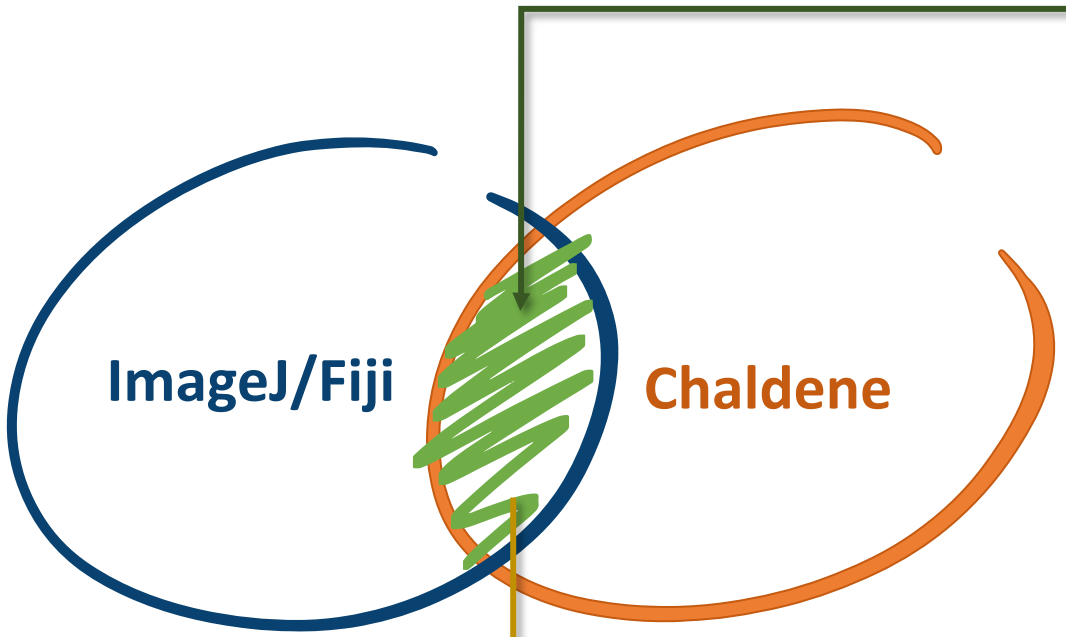
Chaldene

- Python-based Jupyter extension
- × Brand new tool
- ✓ Access to Python libraries
- ✓ Easy to share and modify
  - ✓ Visual automation





Motivation - Integration



We want

ImageJ Function + Chaldene System

= Familiar functions for diverse communities

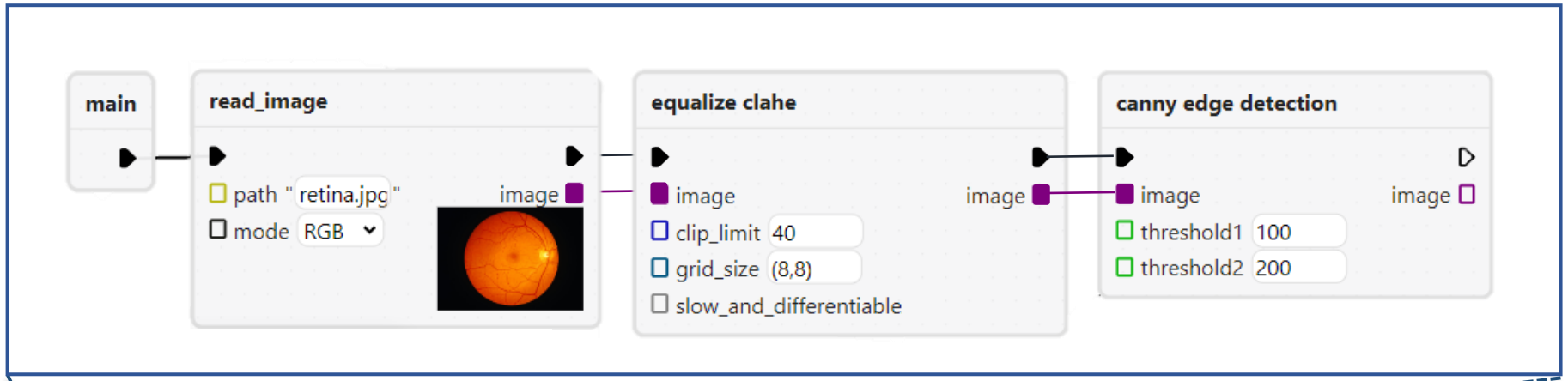
= To expand boundary of ImageJ-confined scientists by other Python libraries

= To achieve **digital sustainability** and **scientific reproducibility** by providing visual automation workflow

Challenge → How do we integrate **Java-based** ImageJ/Fiji into **Python-based** Chaldene?

Methodology – Chaldene in JupyterLab

Frontend View



Backend View

Textual Source Code

```
import cv2 as cv
import kornia as K
from torchvision import io
from torchvision.io import ImageReadMode
n_1_image = io.read_image('retina.jpg', ImageReadMode.RGB)
n_1_image = {
  'value': n_1_image,
  'dataType': 'torch.tensor',
  'metadata': {
    'colorChannel': 'rgb',
    'channelOrder': 'channelFirst',
    'isMiniBatched': False,
    'intensityRange': '0-255',
    'device': 'cpu'
  }
}
n_2_image = K.enhance.equalize_clahe(torch2torch(n_1_image, json.loads('[{"colorChannel"
n_2_image = {
  'value': n_2_image,
  'dataType': 'torch.tensor',
  'metadata': {
    'colorChannel': 'rgb' if torch2torch(n_1_image, json.loads('[{"colorChannel":"rgb","
```

Executed With

Jupyter Python Kernel

A **Jupyter kernel** is a programming language-specific engine that drives code execution in Jupyter notebooks.

Methodology – First Step of Challenge

Challenge

How do we integrate **Java-based** ImageJ/Fiji into **Python-based** Chaldene?

First step

How to run **Java-based** ImageJ in Jupyterlab?

ImageJ Textual Source Code

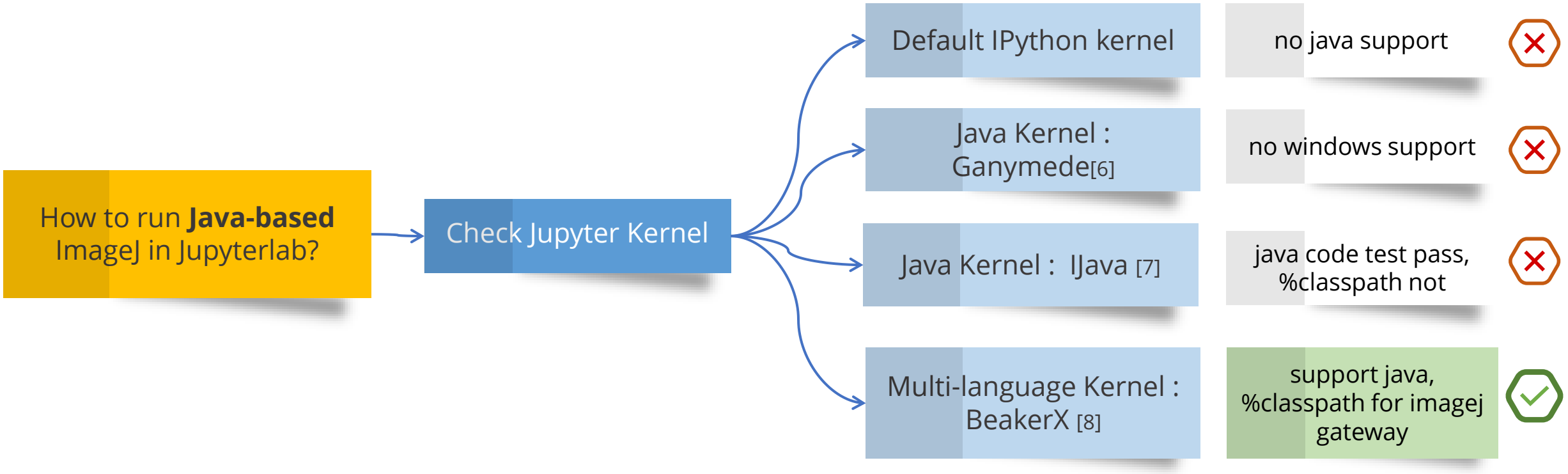
Via **Image Gateway,** we can get access to ImageJ API
- By Loading ImageJ Library from remote Maven repository
- with **%classpath** add mvn net.imagej [9]

Executed With

Jupyter Java Kernel

A **Jupyter kernel** is a programming language-specific engine that drives code execution in Jupyter notebooks.

Methodology – Java Execution in JupyterLab



Progress – ImageJ in JupyterLab via BeakerX

How to run **Java-based** ImageJ in Jupyterlab?

ImageJ Textual Source Code

Via **Image Gateway**

Executed With

Jupyter Java Kernel

BeakerX Kernel

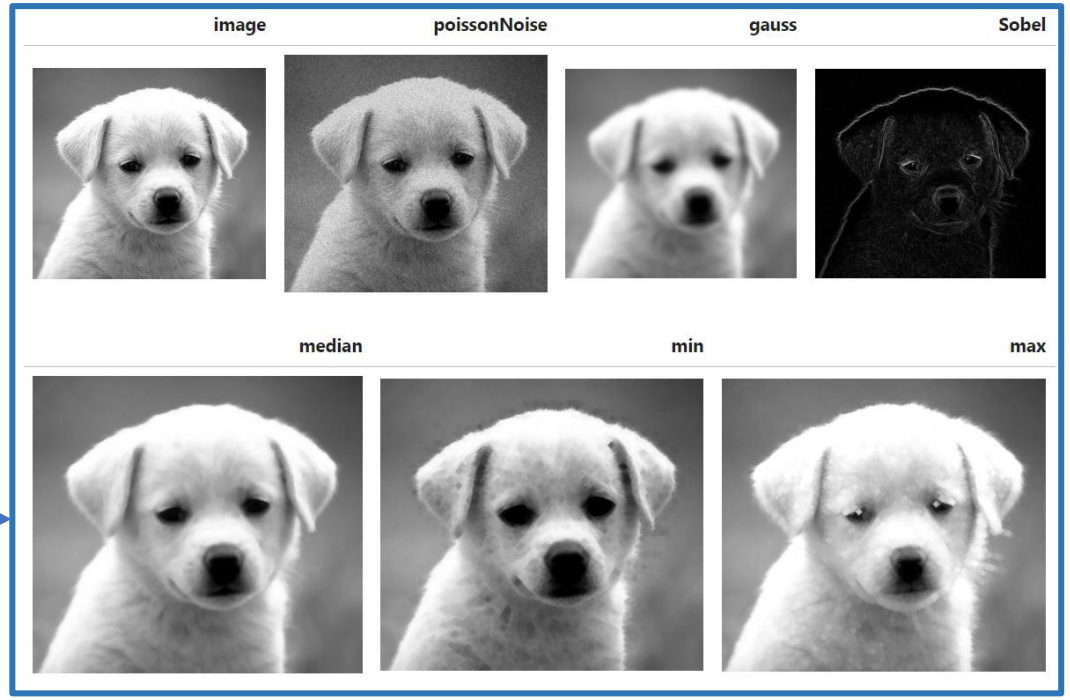


Figure 3 : Image Filtering Result with ImageJ Plugins on Jupyterlab

Methodology – Polyglot Notebook

First step

How to run **Java-based** ImageJ in Jupyterlab?

BeakerX + ImageJ Gateway

- BeakerX supporting
  - Java kernel
  - %classpath
- Get access to ImageJ API via gateway

Next step

How to handle **Java-based** ImageJ and **Python-based** Chaldene in the same Jupyter notebook?

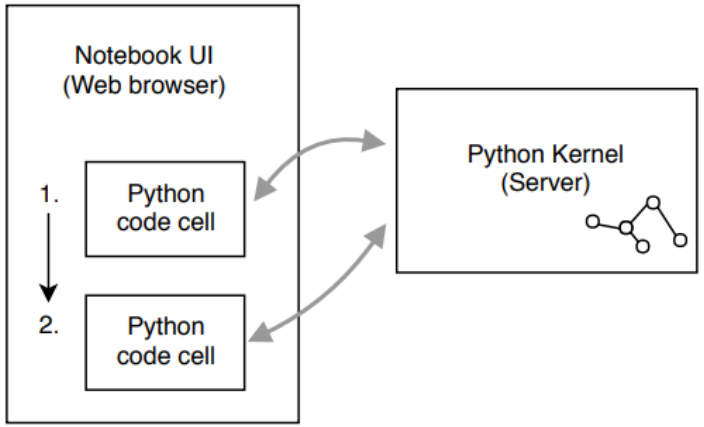
Polyglot Notebook

Support: Java Kernel, Python Kernel — in one notebook

- SoS Notebook [10]
- PolyJus Notebook [11]

Methodology – Polyglot Notebook

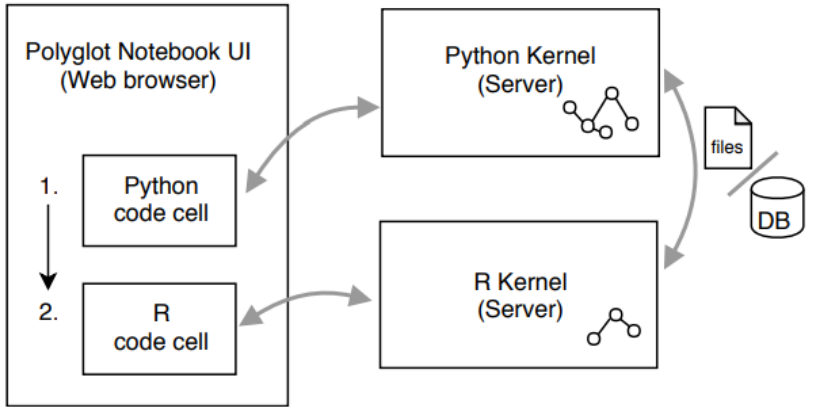
Standard Notebook with Single Kernel e.g. IPython [11]



Standard Notebook

- Single Jupyter execution kernel
- Supports **one** programming language
- All data are stored in one server

Polyglot Notebook with Separate Kernels e.g. SoS [11]

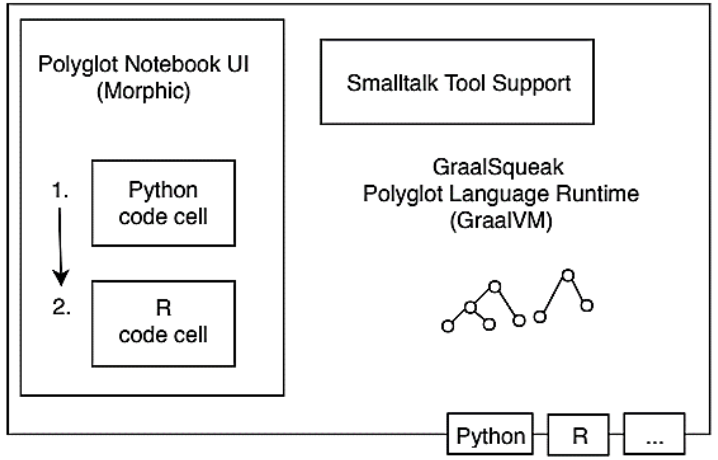


SoS Notebook

- Multiple Jupyter execution kernels
- Supports multi languages
- All data are stored in **additional database**

Changing Kernel would lose all current data

PolyJus Integrated UI and Polyglot Language Runtime[11]



PolyJus Notebook

- One Polyglot kernel based on **GraalVM**
- Supports multi languages
- All data are stored in its **own environment**

Outside Jupyter Environment

Methodology – PyImageJ

How to handle **Java-based** ImageJ and **Python-based** Chaldene in the same Jupyter notebook?

Polyglot Notebook

Support: Java Kernel, Python Kernel — in one notebook

- SoS Notebook [10]
- PolyJus Notebook [11]

Details in Issue 59

New question

How to run **Java-based** ImageJ in Jupyterlab also with **Python Kernel to be compatible with Chaldene**?

PyImageJ

Python Kernel

Wrapping Layer

Java Kernel

Language binding between Python and Java

PyImageJ

- Provide wrapper functions that allow us to run ImageJ with Python
- Provide conversion API for ImageJ image <-> Python Image

Methodology – PyImageJ

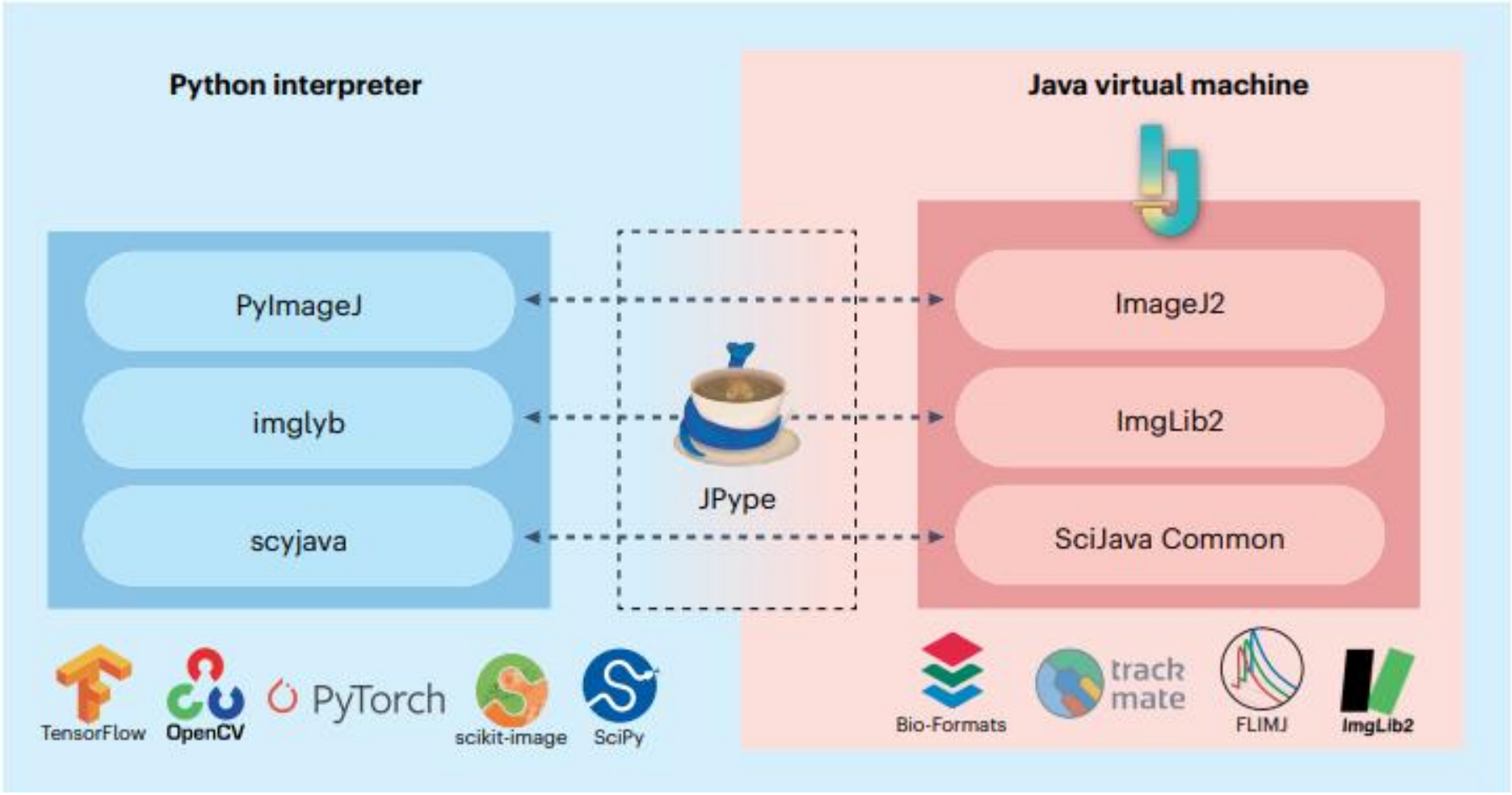


Figure 4 : Architecture of PyImageJ [12]

Progress - Run ImageJ via PyImageJ

Code

Python (pyimagej)

Python Kernel

1.Initialzation of pyimagej

[1]:
```
import imagej

# initialize PyImageJ
ij = imagej.init('sc.fiji:fiji:2.14.0')
print(f"ImageJ version: {ij.getVersion()}")
import sys
print(sys.executable)
```

```
ImageJ version: 2.14.0/1.54f
E:\Users\59803\anaconda3\envs\pyimagej\python.exe
```

2. Import imagej classes by scyjava jimport

[3]:
```
from scyjava import jimport
Runtime = jimport('java.lang.Runtime')
print(Runtime.getRuntime().maxMemory() // (2**20), " MB available to Java")
```

```
3930  MB available to Java
```

[6]:
```
# load test image
dataset1 = ij.io().open('test_image.tif')

# display test image (see the Working with Images for more info)
ij.py.show(dataset1)
```

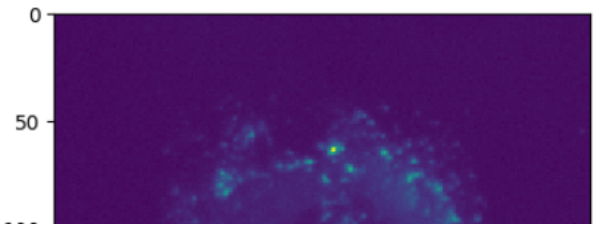


Figure 5: Run ImageJ Functions with Python Kernel

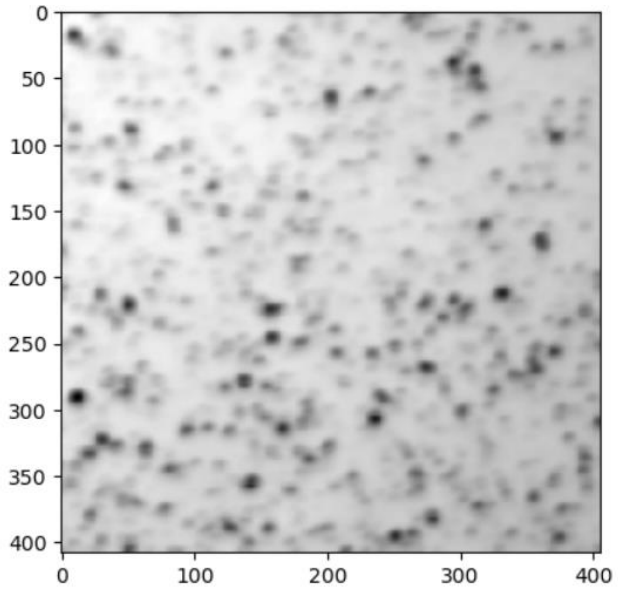
Details in Issue 77

Progress - Run ImageJ via PyImageJ

```
HyperSphereShape = jimport('net.imglib2.algorithm.neighborhood.HyperSphereShape')
radius = HyperSphereShape(2)
```

Import Java Library by JPype

```
ij.op().filter().mean(result, colony_img, radius)
ij.py.show(result,cmap='gray')
```




```
thresholded = ij.op().run("threshold.otsu", result)
ij.py.show(thresholded,cmap='gray')
```

Run ImageJ Plugins

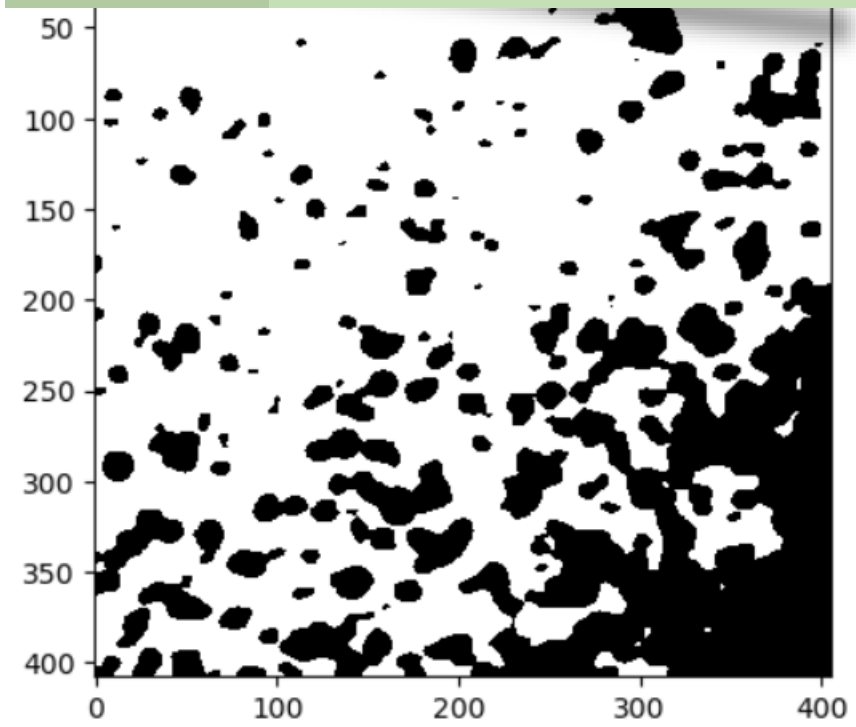
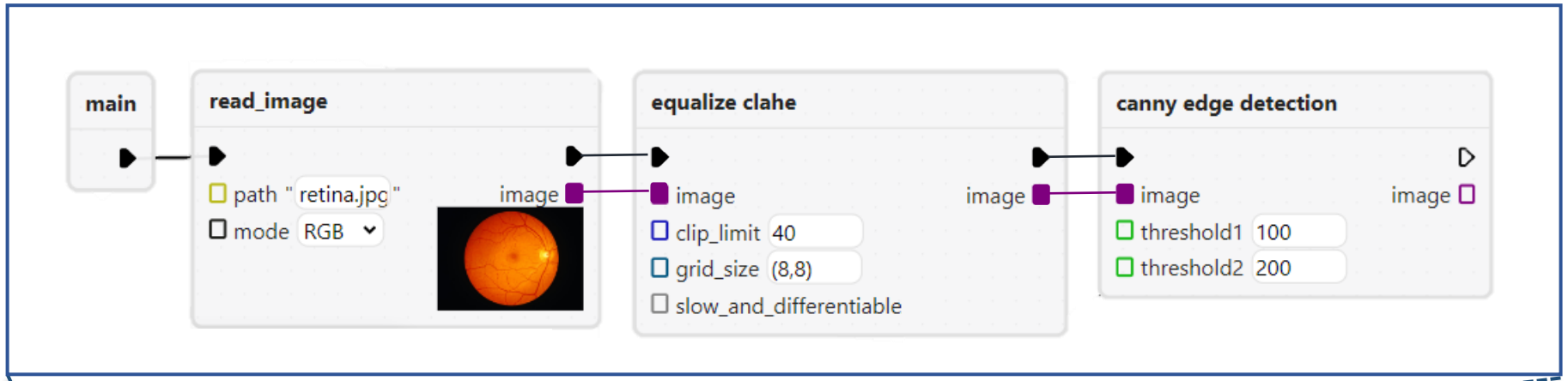


Figure 6: PyImageJ supports most of ImageJ plugins in headless mode

Recap – Chaldene in JupyterLab

Frontend View



Backend View

Textual Source Code

```
import cv2 as cv
import kornia as K
from torchvision import io
from torchvision.io import ImageReadMode
n_1_image = io.read_image('retina.jpg', ImageReadMode.RGB)
n_1_image = {
  'value': n_1_image,
  'dataType': 'torch.tensor',
  'metadata': {
    'colorChannel': 'rgb',
    'channelOrder': 'channelFirst',
    'isMiniBatched': False,
    'intensityRange': '0-255',
    'device': 'cpu'
  }
}
n_2_image = K.enhance.equalize_clahe(torch2torch(n_1_image, json.loads('[{"colorChannel"
n_2_image = {
  'value': n_2_image,
  'dataType': 'torch.tensor',
  'metadata': {
    'colorChannel': 'rgb' if torch2torch(n_1_image, json.loads('[{"colorChannel":"rgb","
```

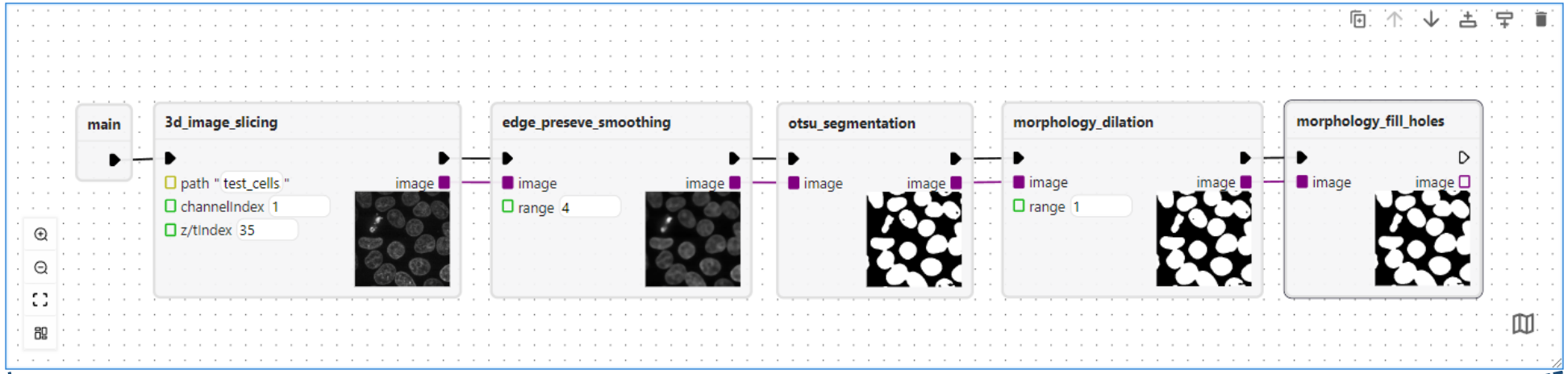
Executed With

Jupyter Python Kernel

A **Jupyter kernel** is a programming language-specific engine that drives code execution in Jupyter notebooks.

Recap – ImageJ in Chaldene

Frontend View



Backend View

ImageJ Textual Source Code

Via **ImageJ Gateway**

Executed With

Jupyter Python Kernel

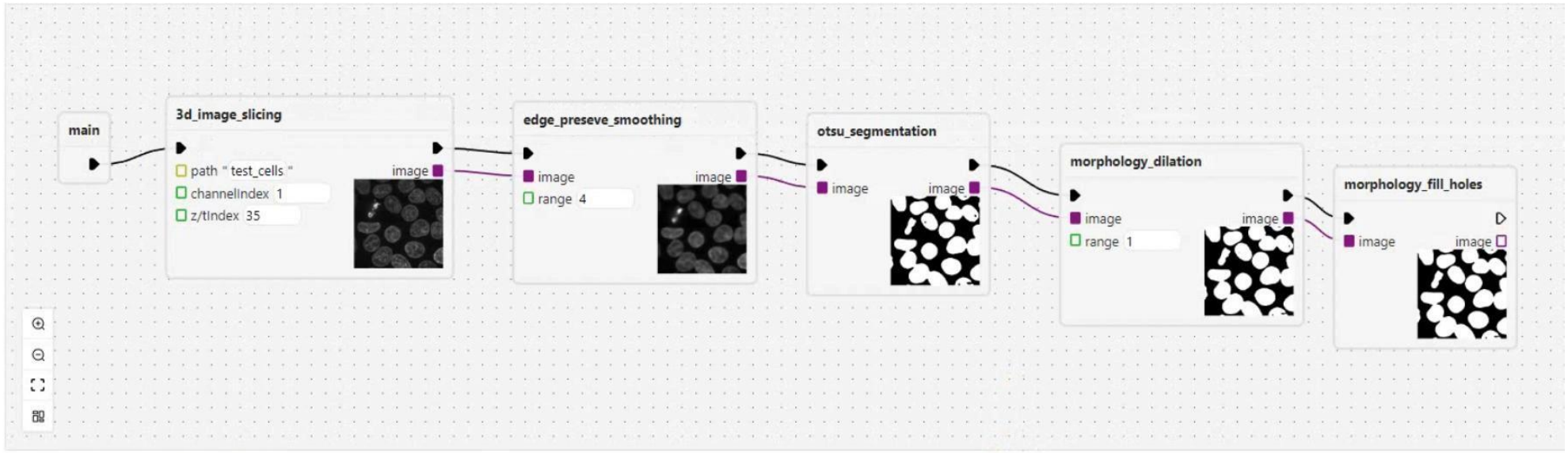
Via PyImageJ's language binding

Progress – ImageJ Workflow in Chaldene

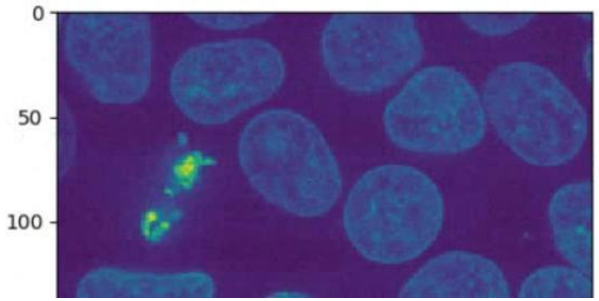
PyImageJ_test1.ipynb | ImageJ_Cell_Segmentatic

Code

[5]:




```
--- Ori 3d Image Information ---
 type: <class 'numpy.ndarray'>
dtype: uint16
shape: (60, 2, 256, 256)
--------------------------------
------ Image After slicing ------
 image slice in channel [1] and Z [35]
```



Details in [Issue 68]

Video 1: ImageJ Cell Segmentation Workflow [13] integrated in Chaldene System

Background – Behind Chaldene Visual Node

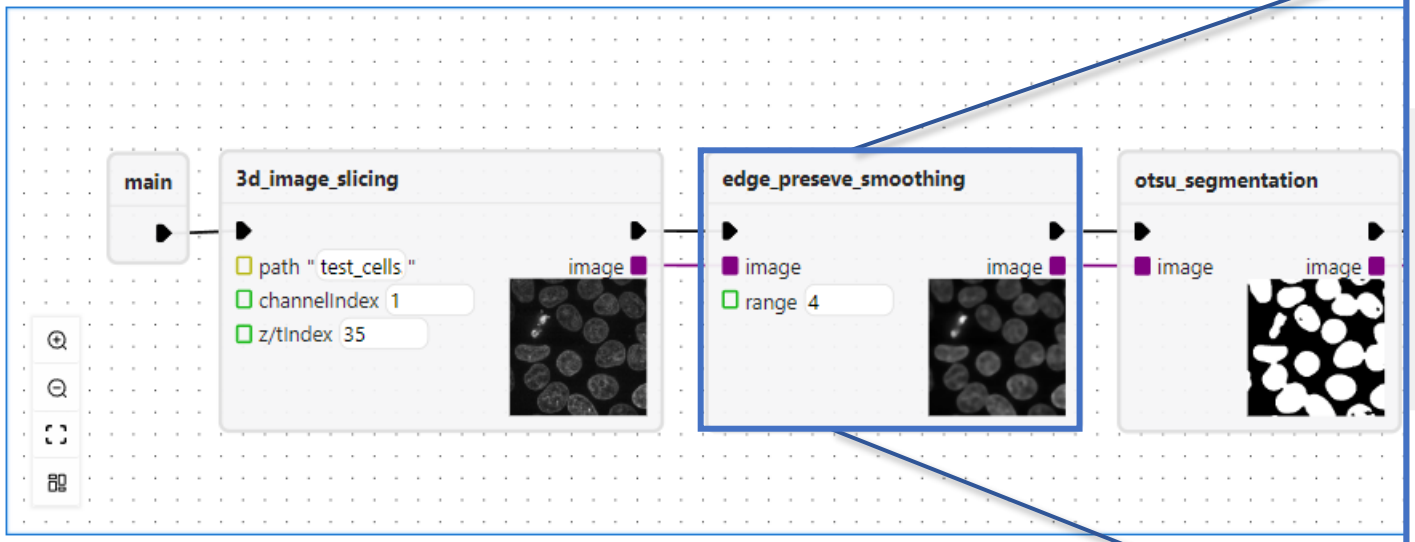


Figure 8: ImageJ Visual Nodes in Chaldene Wor

JSON Node Specification


```
{
  "description": "ImageJ cell segmentation workflow ",
  "enable": true,
  "nodes": {
    "3d_image_slicing": {...},
    "edge_preseve_smoothing": {
      "type": "edge_preseve_smoothing",
      "category": "function",
      "title": "edge_preseve_smoothing",
      "tooltip": "Do edge preserve smoothing on image.",
      "externalImports": "import imagej\nij = imagej.init('sc.fiji:fiji:2.14.0')\nimport scyjava as sj",
      "codeGenerator": "function code(inputs, outputs, node, generator) {\n  const code = `#Get input
       image\\nvp_input_image = ${inputs[1]}\\n#Get Op Name\\nOp_name = \"filter.median\"\\n#Import java
       libararies\\nHyperSphereShape =
       sj.jimport(\"net.imglib2.algorithm.neighborhood.HyperSphereShape\")\\n#Convert input image to Java
       type\\nprocess_image = vp_input_image.get('value')\\njimage = ij.py.to_java(process_image)\\n#Prepare
       result image\\n${outputs[1]} = ij.op().run(\"create.img\",jimage)\\n#Process image by Op\\n${outputs[1]}=
       ij.op().run(Op_name, ij.py.jargs(${outputs[1]}, process_image, HyperSphereShape(5)))\\n#Normalization to
       range (0,255), uint8 \\n${outputs[1]} = ij.py.from_java(${outputs[1]})\\n${outputs[1]} = ((${outputs[1]}
       ${outputs[1]}.min()) / (${outputs[1]}.max() - ${outputs[1]}.min())* 255).astype('uint8')\\n#Display
       result\\ndef image_info(image):\\n    # A handy function to print details of an image object.\\n
       print(\"--- After Edge Preserve Smoothing ---\")\\n    ij.py.show(image)\\n    print(f\" type:
       {type(image)}\")\\n    print(f\"dtype: {image.dtype if hasattr(image, 'dtype') else 'N/A'}\")\\n
       print(f\"shape: {image.shape}\")\\n    print (\"--------------------------------
       \")\\nimage_info(${outputs[1]});\\n#Prepare output metadata\\n${outputs[1]} = {\\n  'value':
       ${outputs[1]},\\n  'dataType': 'numpy.ndarray',\\n  'metadata': {\\n    'colorChannel': 'grayscale',\\n
       'channelOrder': 'channelLast',\\n    'isMiniBatched': False,\\n    'intensityRange': '0-255',\\n
       'device': 'cpu'\\n  }\\n}\\n${outputs[0]}`;\n  return code;\n}",
      "inputs": {
        "execIn": {...},
        "image": {
          "title": "image",
          "dataType": "image",
          "defaultValue": {...},
          "tooltip": "input image."
        },
        "range": {
          "title": "range",
          "dataType": "integer",
          "defaultValue": 4,
          "tooltip": "Desired smooth range "
        }
      },
      "outputs": {
        "execOut": {...},
        "image": {
          "title": "image",
          "dataType": "image",
          "defaultValue": {
            "dataType": "numpy.ndarray"
```


Figure 9: JSON Node Specification behind Visual Nodes

Motivation – Auto Generation for ImageJ Op

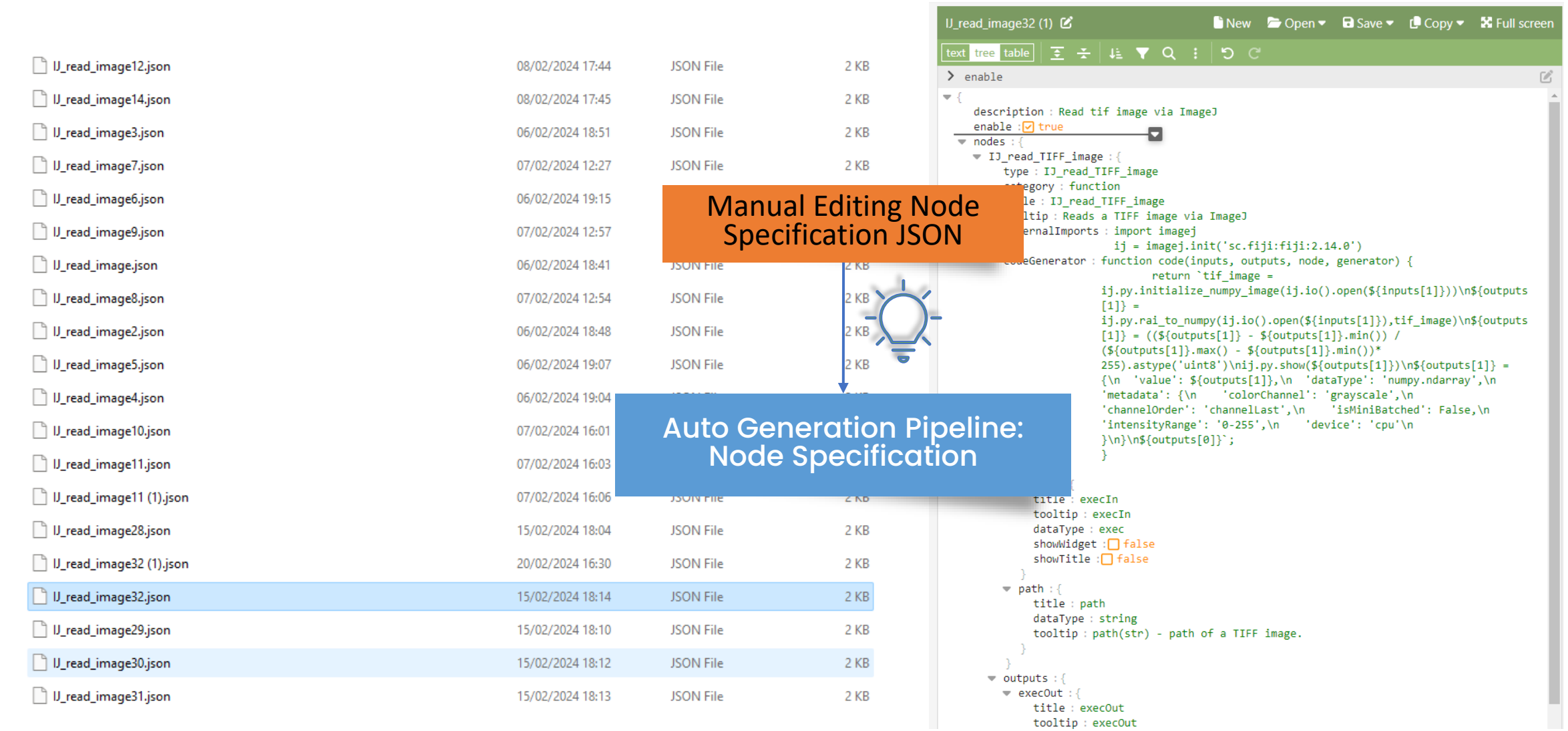
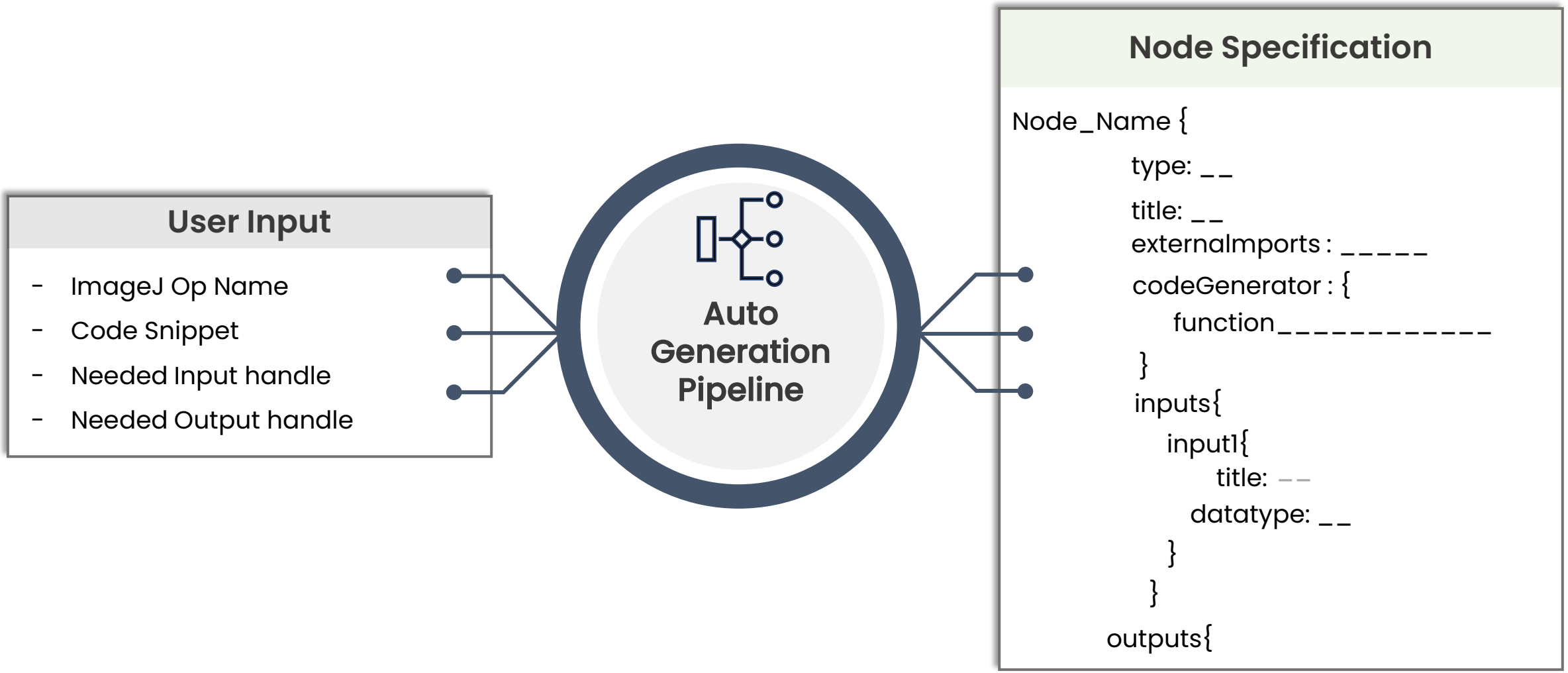
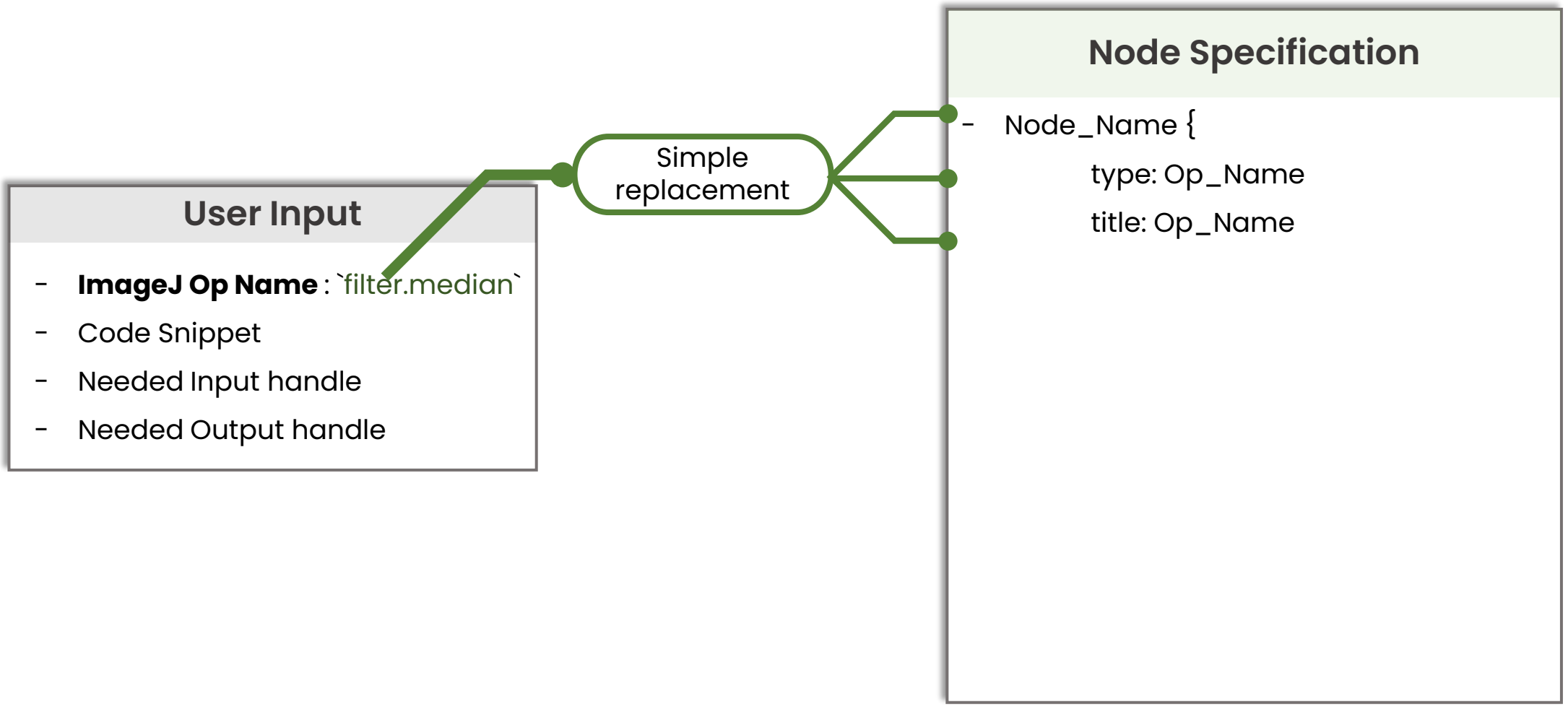


Figure 8: Thirty times of manual editing for one Visual Node Specification JSON file

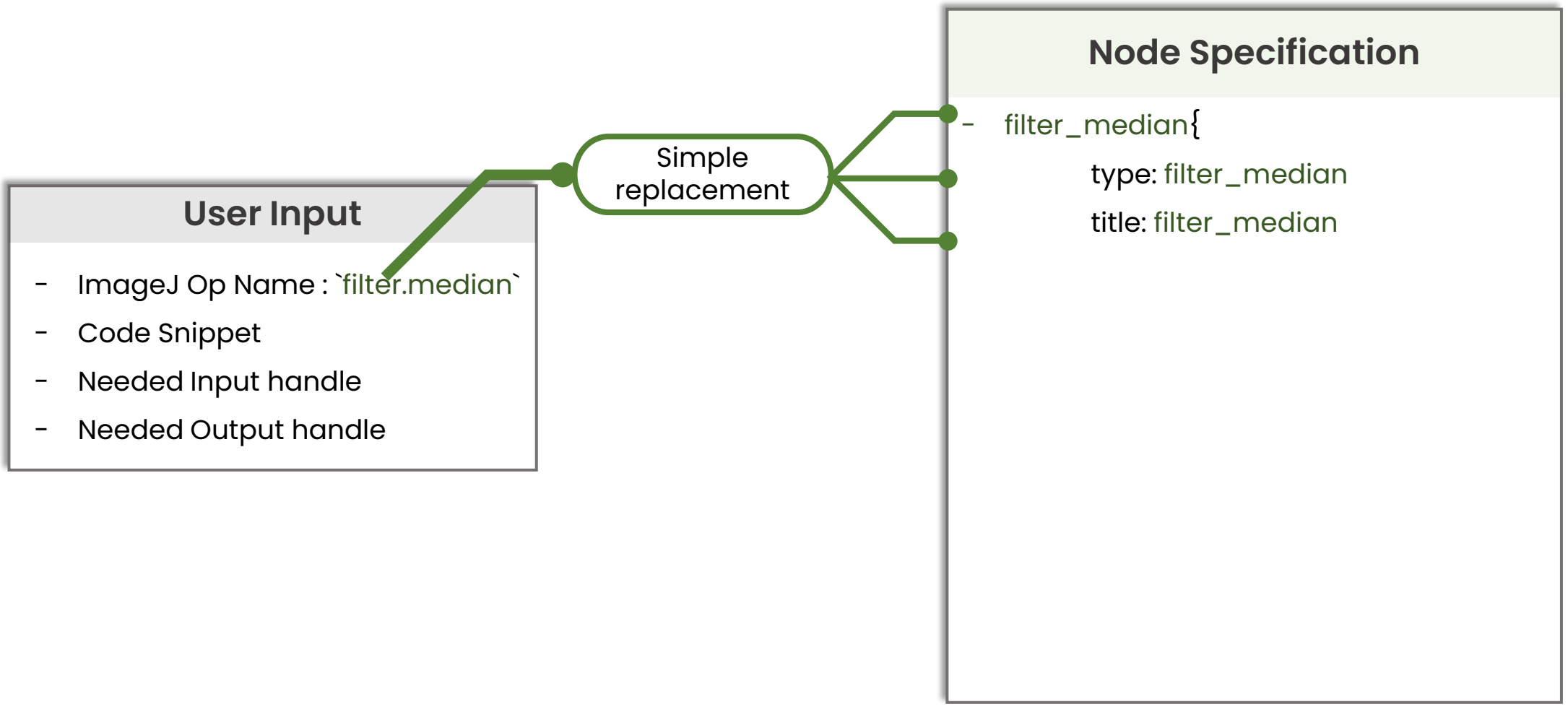
Methodology – Auto Generation for ImageJ Op



Methodology - Pipeline 1 Update Op name



Methodology - Pipeline 1 Update Op name



Methodology – Pipeline 2 Extract Imports

User Input

- ImageJ Op Name
- **Code Snippet**
- Needed Input handle
- Needed Output handle

Sample Code Snippet

```
- import imagej
- import scyjava as sj
- ij = imagej.init('sc.fiji:fiji:2.14.0')
- result_image =
  ij.op().run("create.img",jimage
- result_image= ij.op().run(Op_name,
  ij.py.jargs(result_image,
  process_image,
  HyperSphereShape(range),None))
```

AST code analysis

Node Specification

```
- filter_median{
      type: filter_median
      title: filter_median
      externalImports :
```

Methodology – Pipeline 2 Extract Imports

User Input

- ImageJ Op Name
- **Code Snippet**
- Needed Input handle
- Needed Output handle

Sample Code Snippet

- result_image = ij.op().run("create.img",jimage
- result_image= ij.op().run(Op_name, ij.py.jargs(result_image, process_image, HyperSphereShape(range),None))

AST code analysis

Node Specification

- filter_median{

  type: filter_median

  title: filter_median

  externalImports :
  - import imagej
  - import scyjava as sj
  - ij = imagej.init('sc.fiji:fiji:2.14.0')

Methodology – Pipeline 3 Format codeGenerator

User Input

- ImageJ Op Name
- **Code Snippet**
- **Needed Input handle**
- **Needed Output handle**

Sample Code Snippet

- result_image = ij.op().run("create.img",jimage
- result_image= ij.op().run(Op_name, ij.py.jargs(result_image, process_image, HyperSphereShape(range),None))

Specific code formatting

Node Specification

- filter_median{

  type: filter_median

  title: filter_median

  externalImports :
  - import imagej
  - import scyjava as sj
  - ij = imagej.init('sc.fiji:fiji:2.14.0')

  codeGenerator :

```
function codeGenerator(inputs,outputs,code){
return `
${output1} = _ _ _(${input1},${input2})/n`
}
```

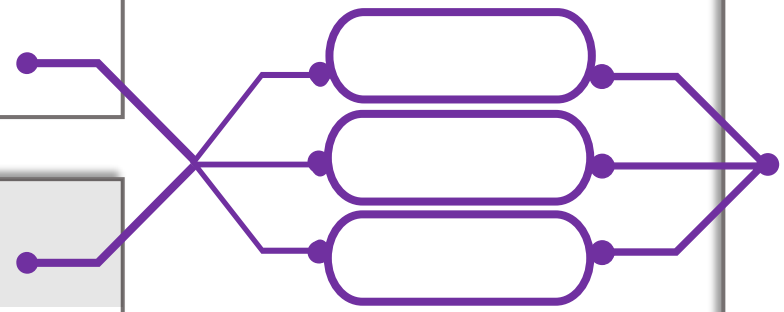
Methodology – Pipeline 4 Update Input/Output

User Input

- **ImageJ Op Name**
- **Code Snippet**
- Needed Input handle
- Needed Output handle

Static Code Snippet

- import imagej
- import scyjava as sj
- ij = imagej.init('sc.fiji:fiji:2.14.0')
- result_image = ij.op().run("create.img",jimage
- result_image= ij.op().run(Op_name, ij.py.jargs(result_image, process_image, HyperSphereShape(range),None))



Node Specification

- filter_median{

type: filter_median

title: filter_median

externalImports :

- import imagej
- import scyjava as sj
- ij = imagej.init('sc.fiji:fiji:2.14.0')

codeGenerator :

```
function codeGenerator(inputs,outputs,code){
return `
${output1} = ___(${input1},${input2})/n`
}
```

inputs :{

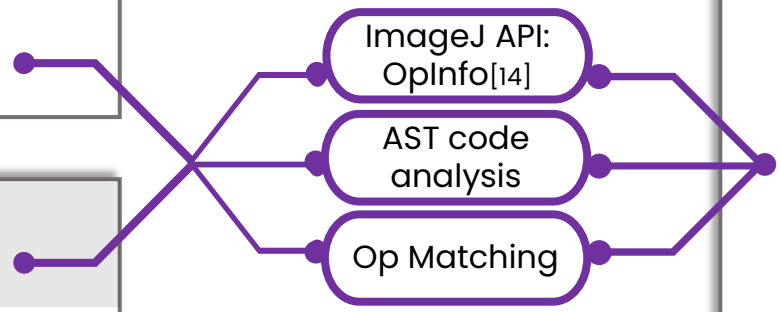
Methodology – Pipeline 4 Update Input/Output

User Input

- **ImageJ Op Name**
- **Code Snippet**
- Needed Input handle
- Needed Output handle

Static Code Snippet

- import imagej
- import scyjava as sj
- ij = imagej.init('sc.fiji:fiji:2.14.0')
- result_image = ij.op().run("create.img",jimage
- result_image= ij.op().run(Op_name, ij.py.jargs(result_image, process_image, HyperSphereShape(range),None))



Node Specification

- filter_median{

type: filter_median

title: filter_median

externalImports :

- import imagej
- import scyjava as sj
- ij = imagej.init('sc.fiji:fiji:2.14.0')

codeGenerator :

```
function codeGenerator(inputs,outputs,code){
return `
${output1} = ___(${input1},${input2})/n`
}
```

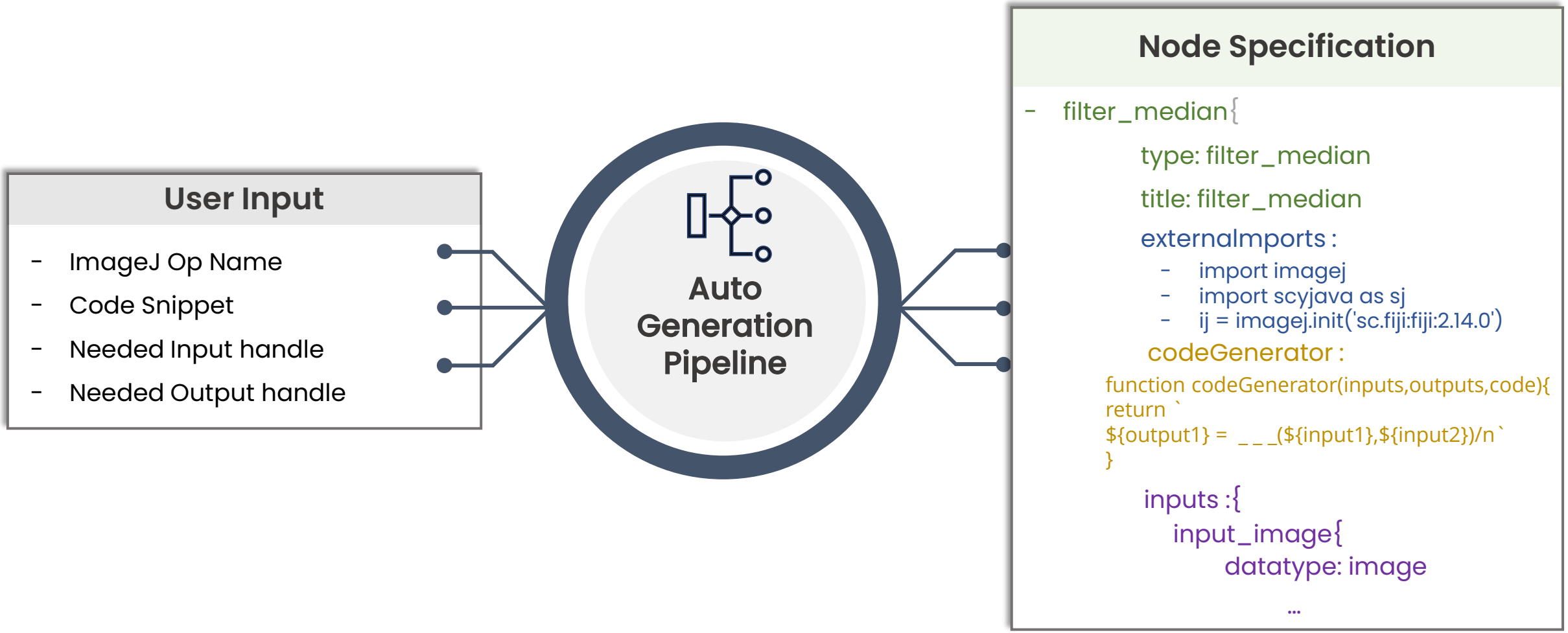
inputs :{

input_image{

datatype: image

...

Methodology – Auto Generation for ImageJ



Progress – Live Generation

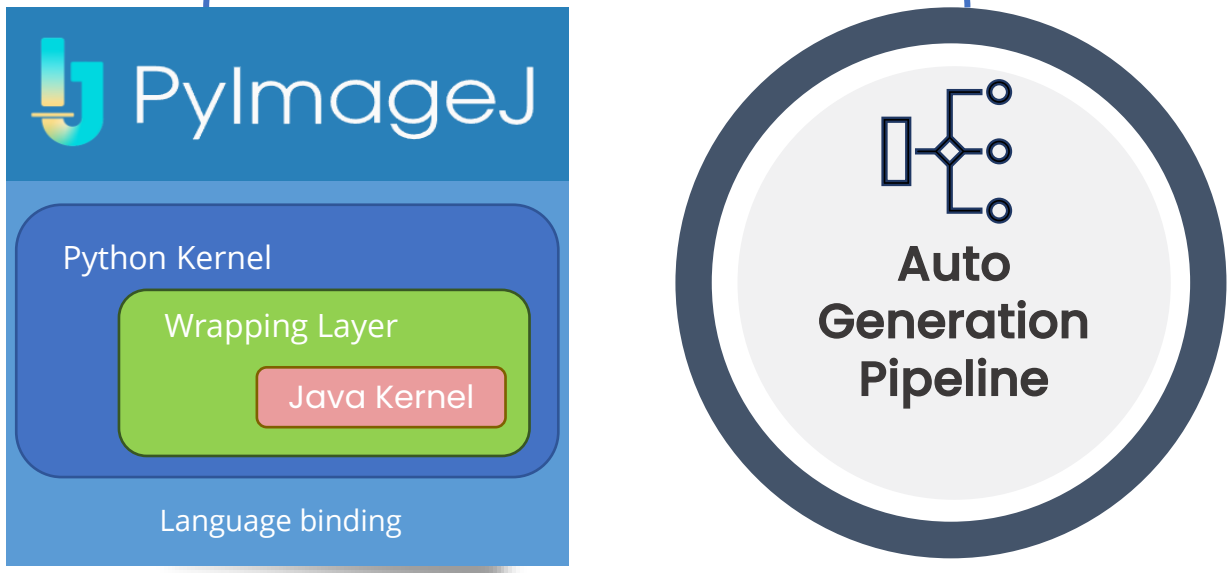
ImageJ Op Node Auto-Generation

Details in [Issue 72]

Conclusion

Tackled challenge

How do we integrate Java-based ImageJ/Fiji into Python-based Chaldene?



We get

ImageJ Function + Chaldene System

= Familiar functions for diverse communities

= To expand boundary of ImageJ-confined scientists by other Python libraries

= To achieve **digital sustainability** and **scientific reproducibility** by providing visual automation workflow

Next Step

- **Enhancement of Auto-Op-generation Pipeline**
  - Wrap up generation pipeline with a special GUI in Chaldene
  - From current one Node generation to one Workflow generation
  - Or if you also have some good ideas to share :)

References

- [1] Schindelin, Johannes, et al. "Fiji: an open-source platform for biological-image analysis." Nature methods 9.7 (2012): 676-682.
- [2] “AstroImageJ: A Simple and Powerful Tool for Astronomical Image Analysis and Precise Photometry”. Astrobites, by Gudmundur Stefansson, 15.03.2024 https://astrobites.org/2016/04/15/astroimagej-a-simple-and-powerful-tool-for-astronomical-image-analysis-and-precise-photometry
- [3] Haase, R. Fiji: Image analysis with the head in clouds. ImageJ API-beating: ImgLib2, ImageJ2 and the Big-Data Viewer. Training School, Luxembourg. 2019.
- [4] Chen, Fei, et al. "Chaldene: Towards Visual Programming Image Processing in Jupyter Notebooks." 2022 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC). IEEE, 2022.
- [5] “Introduction into Macro Programming” ImageJ. 19.03.2024. https://imagej.net/scripting/macro#the-recorder
- [6] “*Ganymede Java Kernel*”. GitHub, owned by Allen Ball, 14.03.2024. https://github.com/allen-ball/ganymede
- [7] “*IJava Java Kernel*”. GitHub, owned by SpencerPark, 14.03.2024. https://github.com/SpencerPark/IJava
- [8] “BeakerX Multi-language Kernel”. GitHub, owned by twosigma, 14.03.2024. https://github.com/twosigma/beakerx
- [9] “ImageJ Gateway”. GitHub, owned by ImageJ, 14.03.2024. https://github.com/imagej/tutorials/blob/master/notebooks/1-Using-ImageJ/1-Fundamentals.ipynb
- [10] Peng, Bo, et al. "SoS Notebook: an interactive multi-language data analysis environment." Bioinformatics 34.21 (2018): 3768-3770.
- [11] Niephaus, Fabio, et al. "PolyJuS: a Squeak/Smalltalk-based polyglot notebook system for the GraalVM." Companion Proceedings of the 3rd International Conference on the Art, Science, and Engineering of Programming. 2019.
- [12] Rueden, Curtis T., et al. "PyImageJ: A library for integrating ImageJ and Python." Nature methods 19.11 (2022): 1326-1327.
- [13] "*Classic Segmentation Workflow with ImageJ2*." PyImageJ.14.03.2024. https://py.imagej.net/en/latest/Classic-Segmentation.html#segmentation-workflow-with-imagej2
- [14] Figge, Marc, Ruman Gerst, and Zoltan Cseresnyes. "JIPipe: Visual batch processing for ImageJ." 2022.

Thank you very much!
Any questions?