



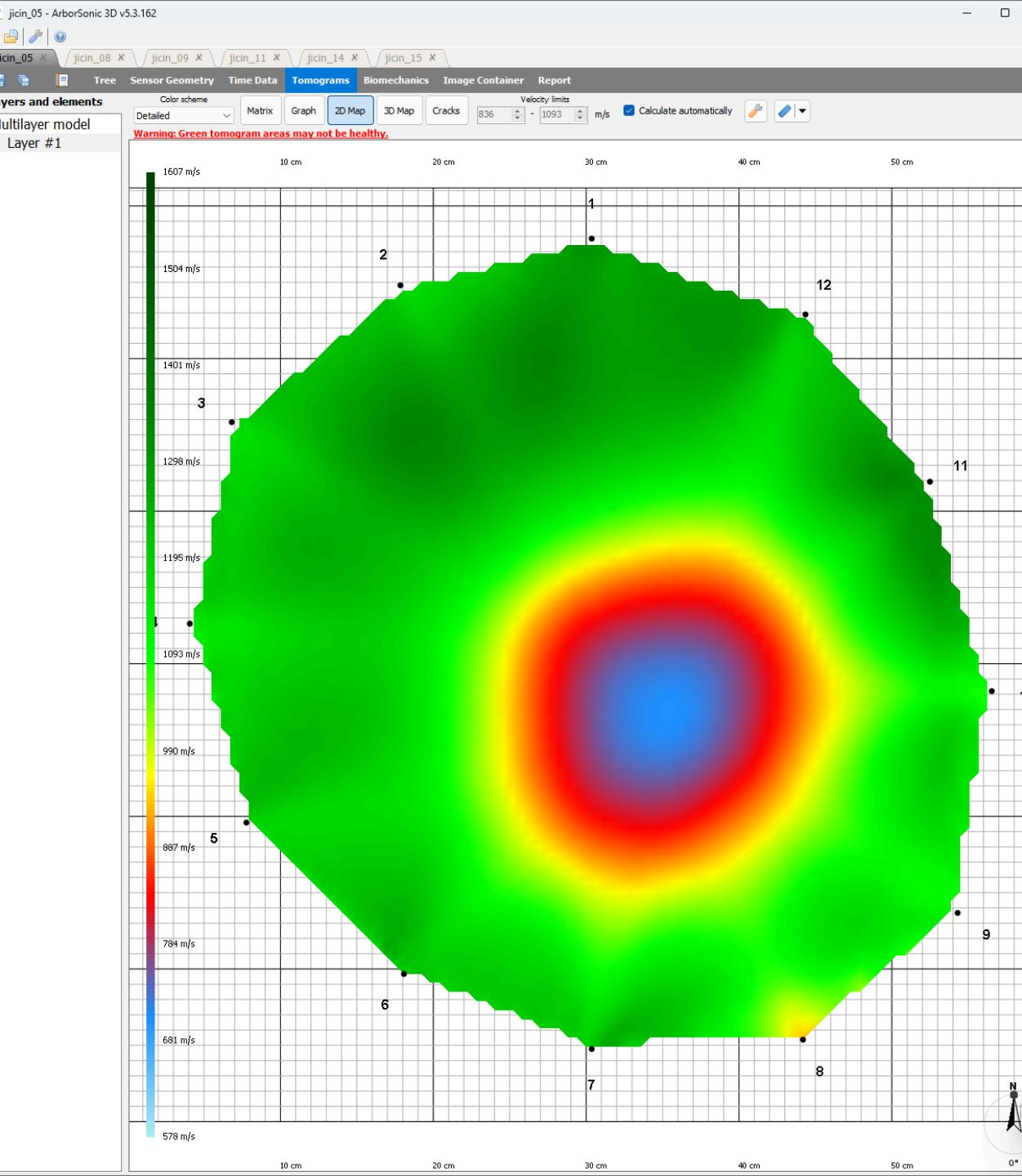
# Resistograph meets tomograph

Robert Mařík & Valentino Cristini  
Mendel University in Brno



# Content of the talk

- Resistograph and tomograph:  
strengths and limitations
- Combined approach: a Python  
library to merge data from  
both devices
- Vibe coding in 2025 (ChatGPT)
- Code sharing in 2025 (Docker)

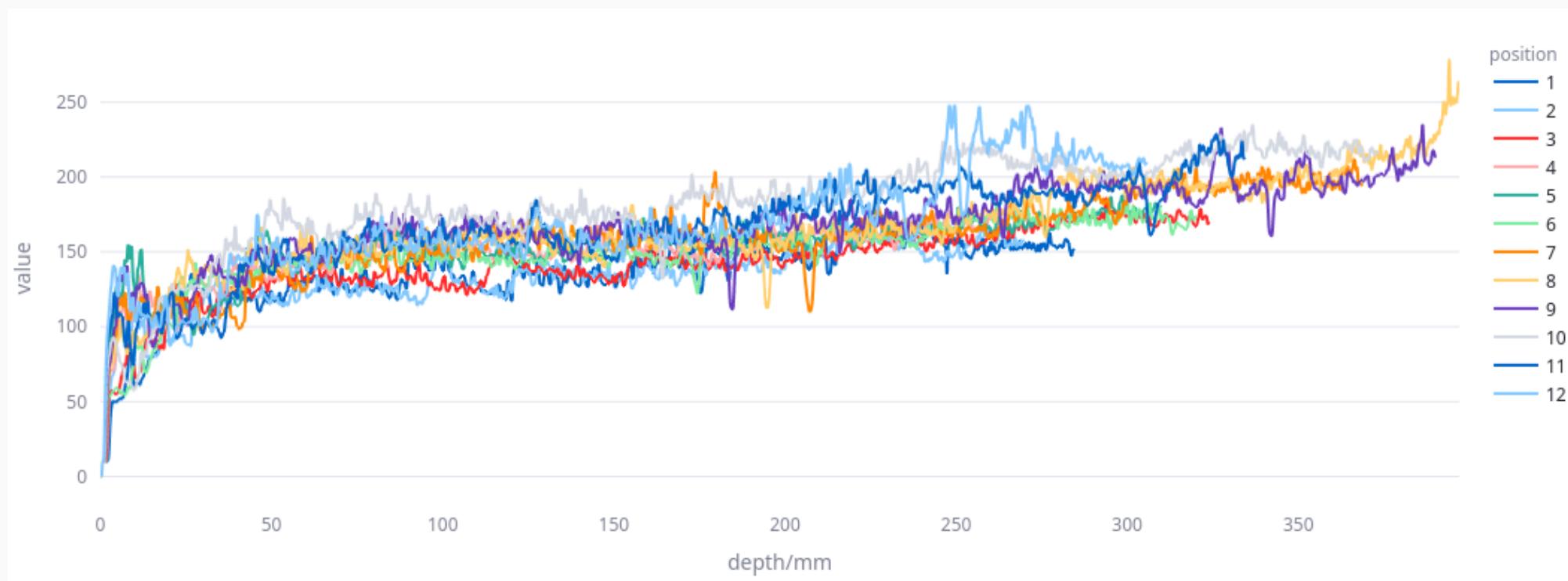


# Tomograph

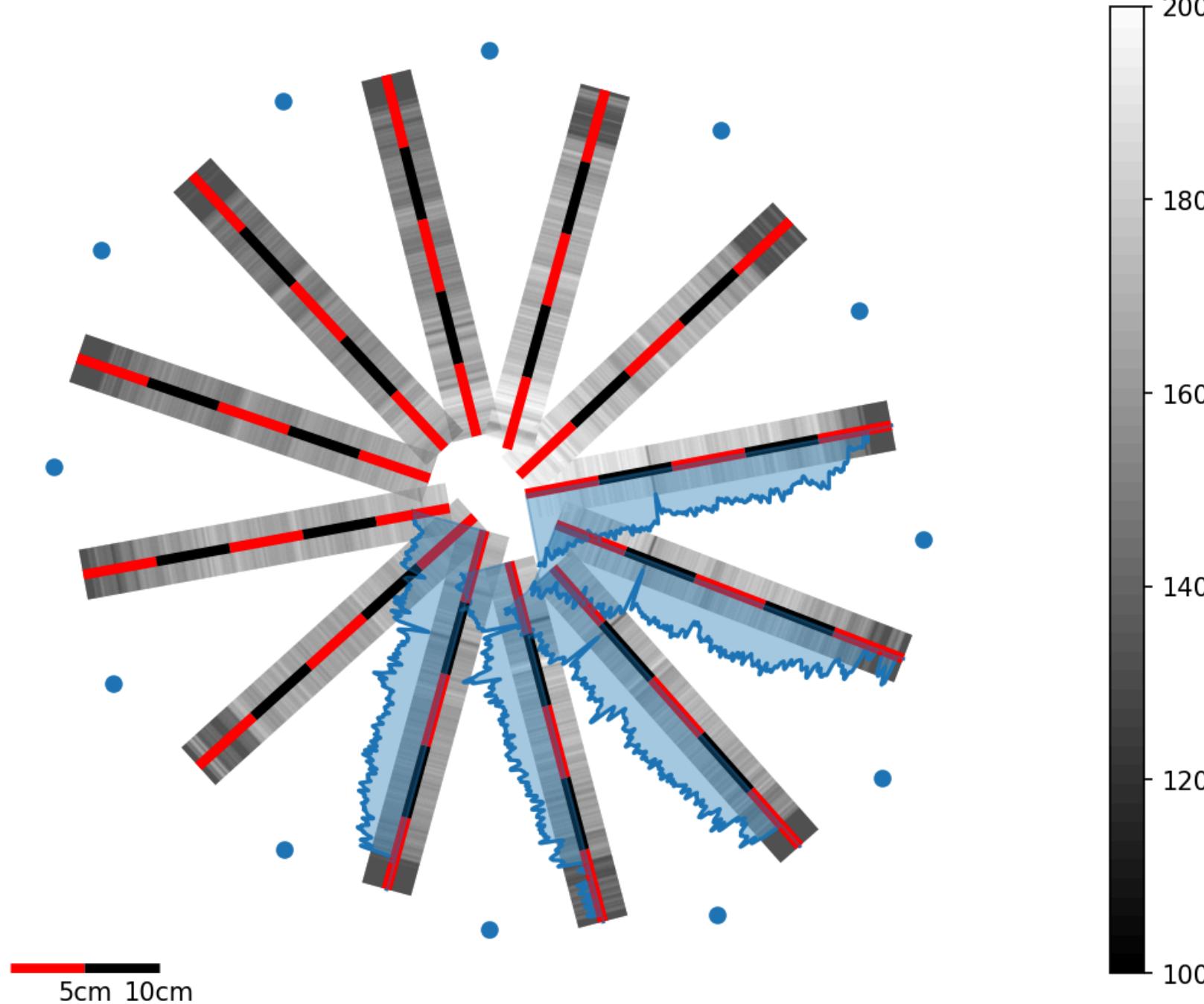
- fast and reliable tool for stem inspection
- global information from the whole cross section
- shows the size and shape of the internal defects
- cracks are reported as cavities

# Resistograph

- scans the power required to microdrilling at given speed
- measures mechanical properties of the material
- local information

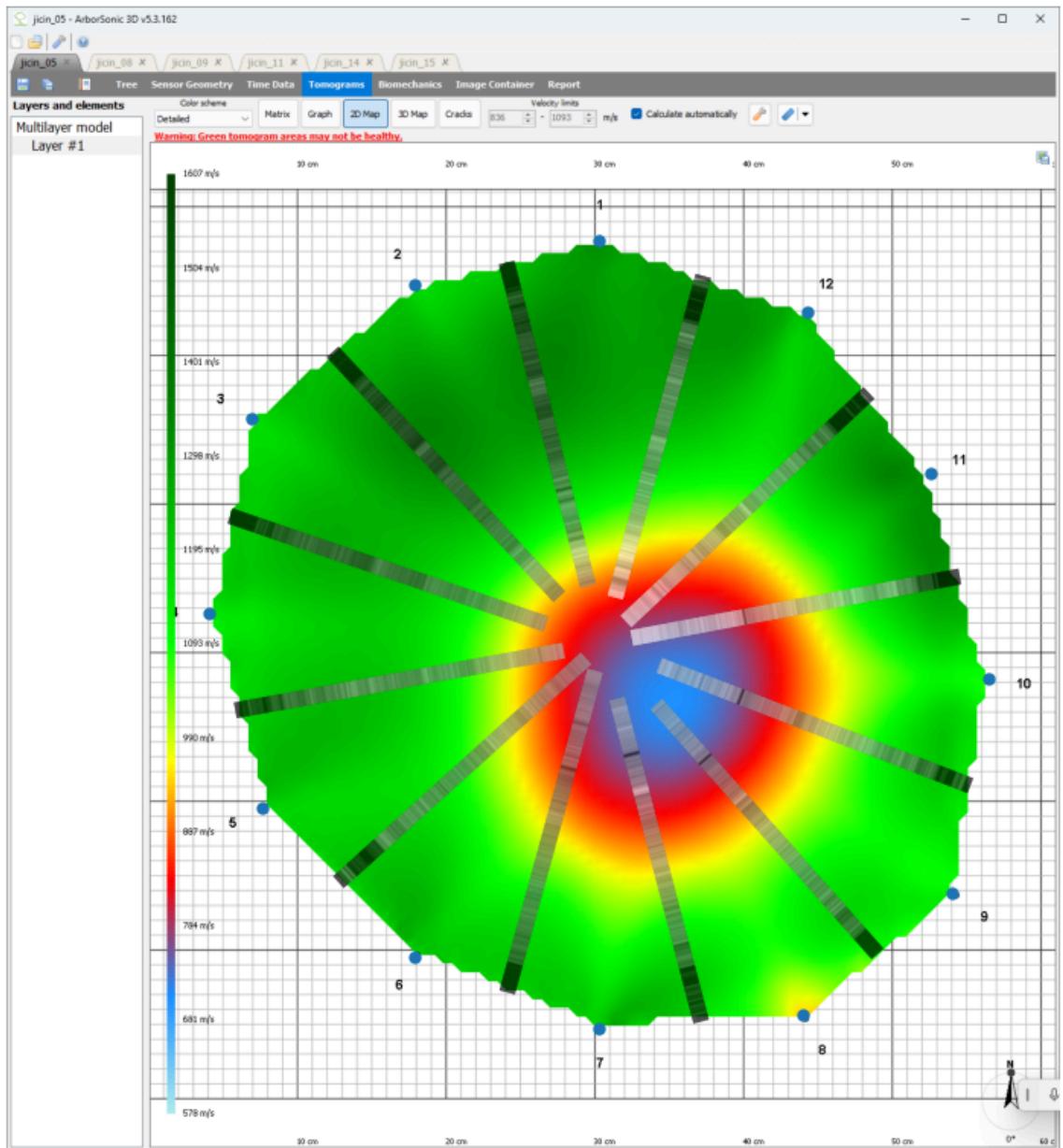


## Resistograph Data Visualization in 2D plane



## Merge data I

- Transform resistograph data to 2D geometry of the cross section
- Visualize the data in the new geometry



# Merge data II

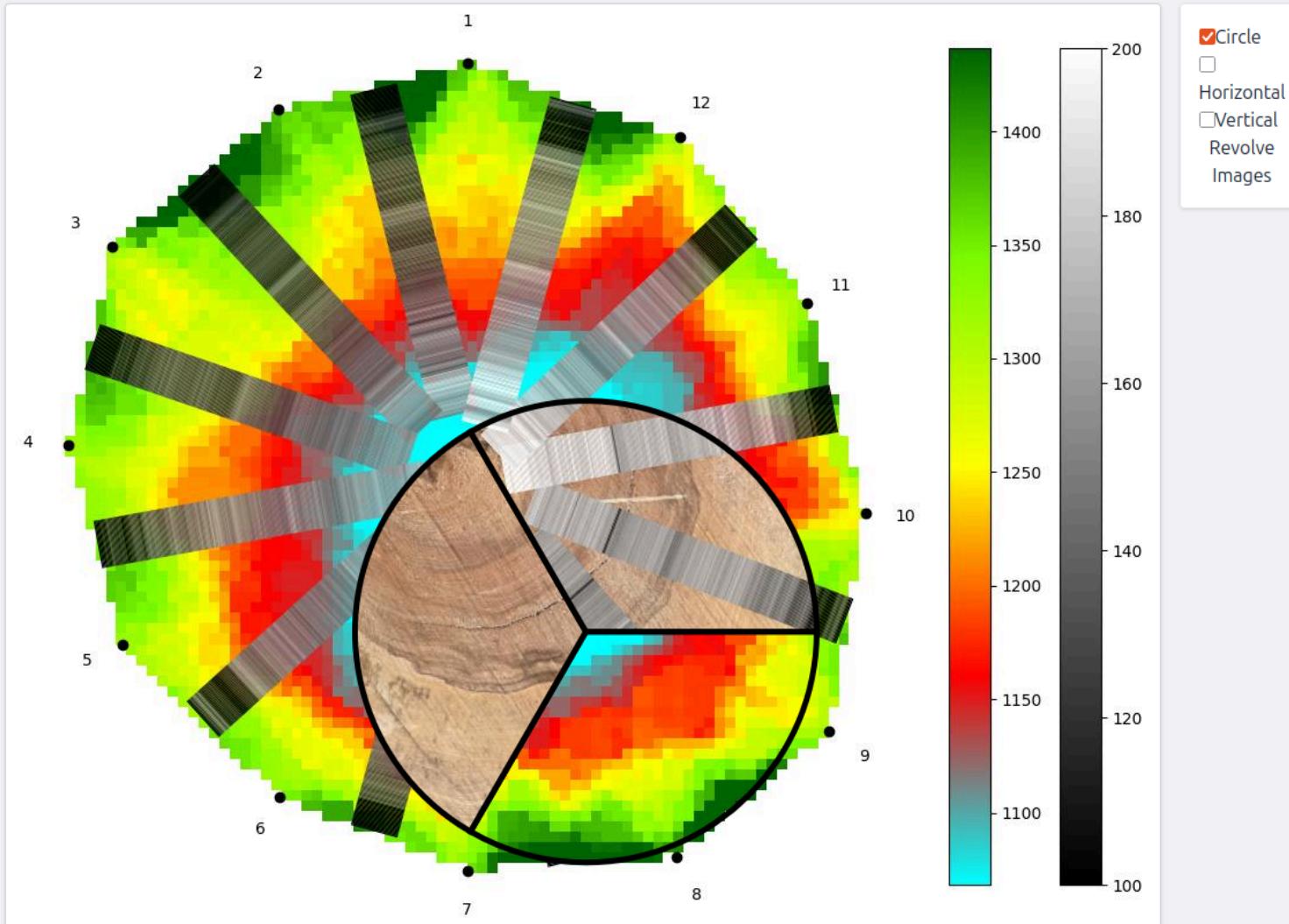
- merge resistograph data with tomograph data
- visualize the merged data
- look for short or long decreases in resistograph data. This indicates cracks and cavities, respectively

## When resistograph meets tomograph

The demo of overlays of four images. See [the repository](#) for the code.

- Tomogram
- Tomogram with resistograph data
- Section photo
- Section photo with resistograph data

You can move the mouse over the image to reveal the other layers or click the image to switch layers.



# Python library

- language widely used in scientific data processing
- many libraries for data processing and visualization
- easy to automate, scale, modify, share and reuse
- easy to integrate with other tools

```

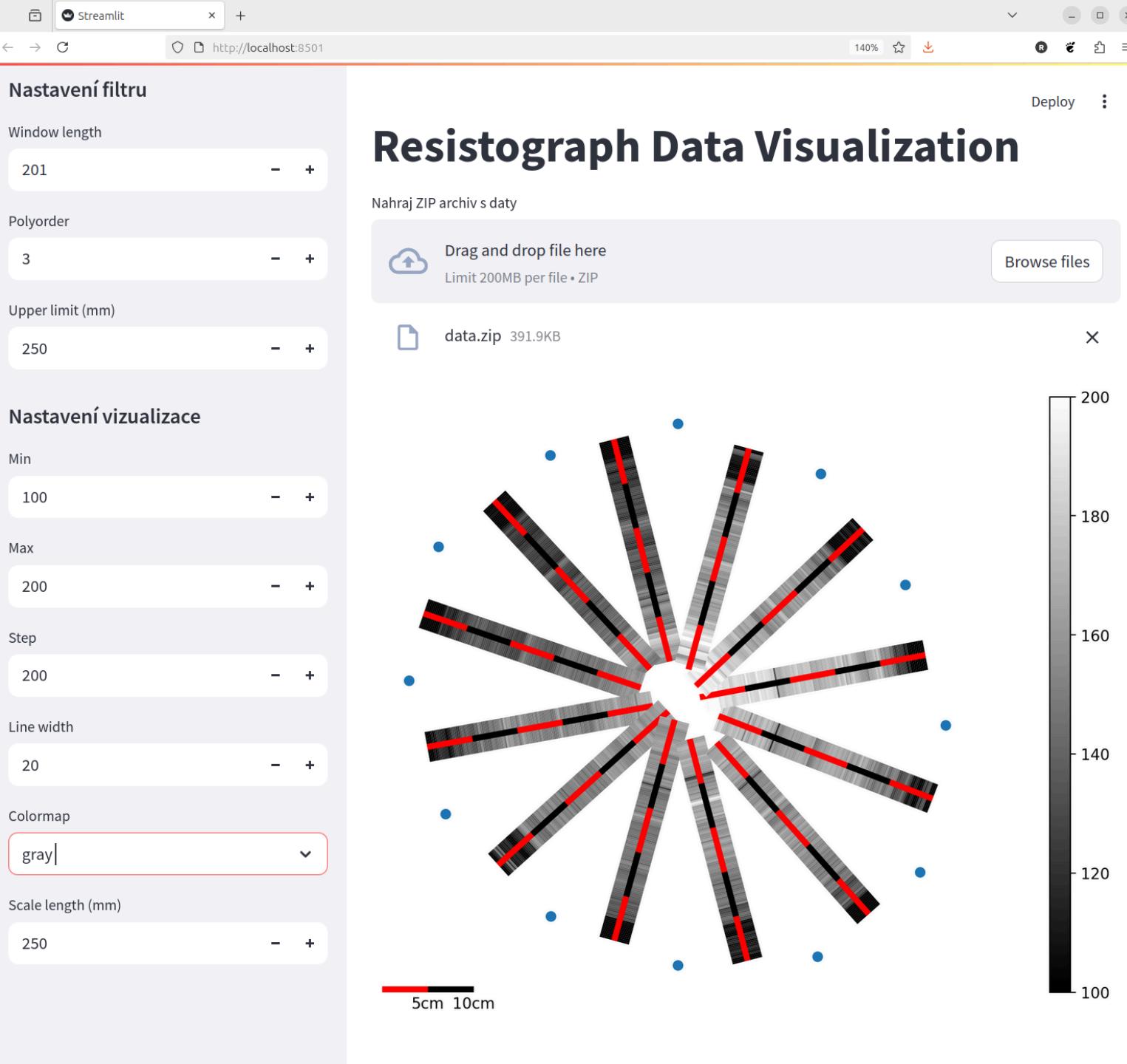
4 # See LICENSE file or https://creativecommons.org/licenses/by/4.0/
Run Cell | Run Below | Debug Cell
5 #%%
6 """
7 This script visualizes resistograph data on a tomogram.
8 It processes resistograph data files and node coordinates to generate a plot
9 with resistograph data overlaid on a tomographic representation.
10
11 Configuration and validation are handled via Pydantic models.
12 """
Run Cell | Run Above | Debug Cell
13 #%%
14 import pandas as pd
15 import numpy as np
16 import matplotlib.pyplot as plt
17 import glob
18 from scipy.signal import savgol_filter
19 import logging
20 from matplotlib.collections import LineCollection
21 from matplotlib.transforms import Affine2D
22
23 # --- NEW: importy pro konfiguraci ---
24 from pydantic import BaseModel, Field, PositiveInt, DirectoryPath, model_validator
25 from typing import List, Optional
26
27 # Logging configuration
28 logging.basicConfig(level=logging.WARNING, format='%(levelname)s - %(message)s')
29
30 # --- NEW: Pydantic models for configuration ---
31 class FilterSettings(BaseModel):
32     window_length: PositiveInt = Field(201, description="Window length for Savitzky-Golay filter")
33     polyorder: int = Field(3, description="Polynomial order for filter")
34     upper_limit: int = Field(250, description="Maximum depth in mm")
35
36     @model_validator(mode="after")
37     def check_polyorder_vs_window(self):
38         if self.polyorder >= self.window_length:
39             raise ValueError("polyorder must be smaller than window_length")
40         return self
41
42
43 class PlotSettings(BaseModel):
44     min: int = Field(100, description="Minimum value for color normalization")
45     max: int = Field(200, description="Maximum value for color normalization")
46     step: int = Field(200, description="Step for downsampling")
47     linewidth: int = Field(20, description="Line width")
48     cmap: str = Field("gray", description="Matplotlib colormap")
49
50     @model_validator(mode="after")

```

# Python library

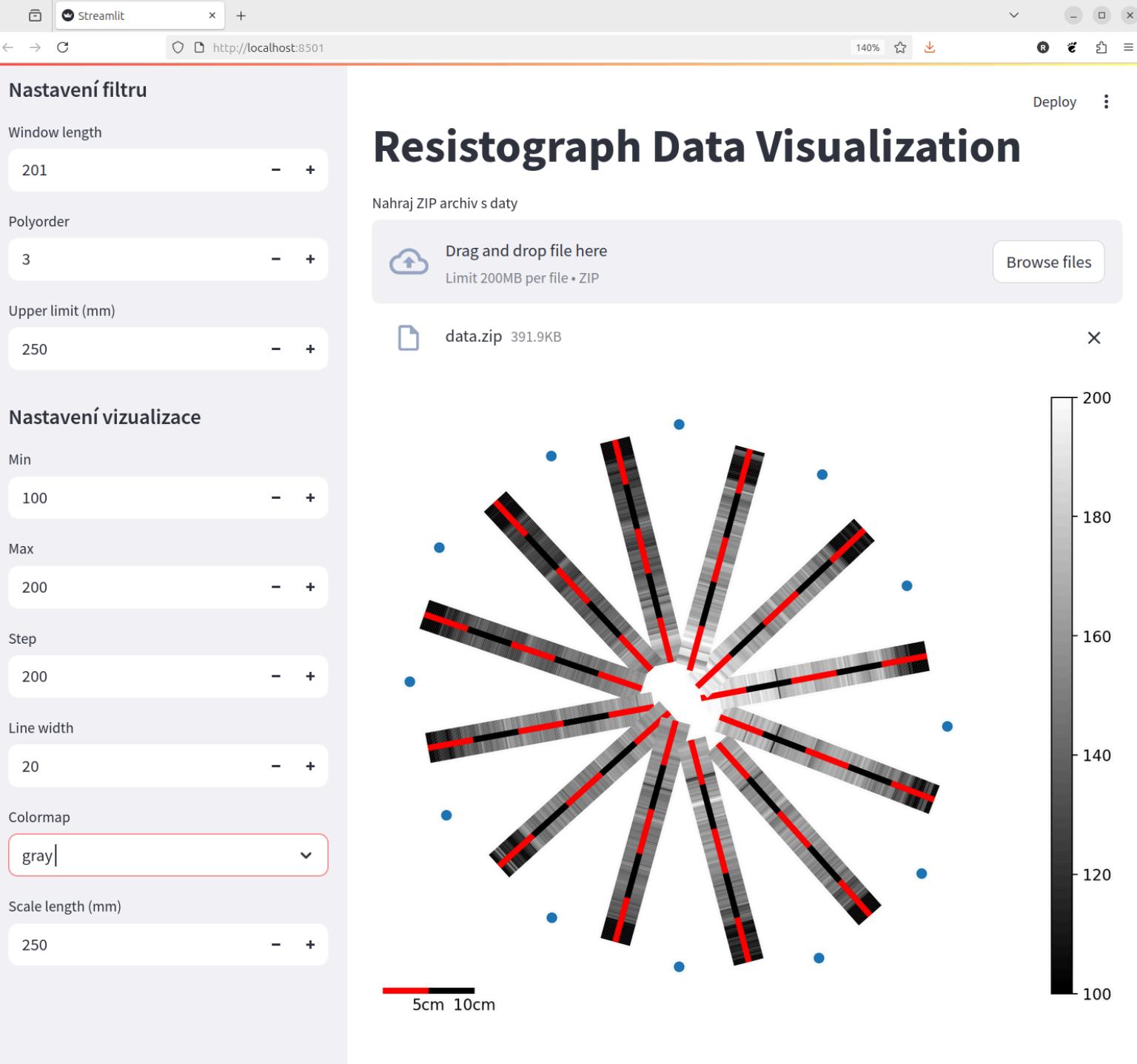
## Limitations

- requires programming skills
- requires installation of Python, Python IDE and libraries
- no GUI



# Streamlit

- library for building web apps
- requires minimal code
- interactive widgets for user input
- real-time updates
- widely used in data science and machine learning, in industry and academia



# Vibe coding

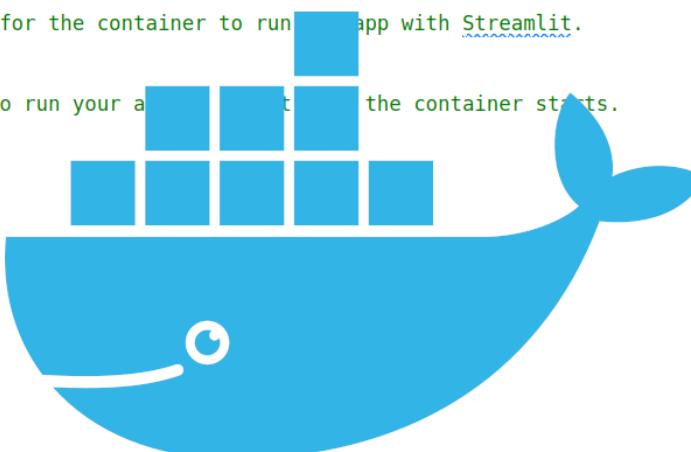
- ChatGPT 5 on August 2025
- web app in two prompts

Mam nasledujici knihovnu. Napis streamlit program, ktery umozni nahrat zazipovany adresar s daty a spusti na nem prikazy odpovidajici main funkci. Vystup se zobrazí.

OK. V levem panelu chci mit moznost menit prednastavene volby.

```
compose.yml > ...
  ▷ Run All Services
1 services:
2   ▷ Run Service
3     resisto:
4       network_mode: bridge
5       working_dir: /app/app
6       ports:
7         - 8501:8501
8       image: resisto:latest
9       build: .
```

```
Dockerfile 1 ×
Dockerfile > ...
1 # This sets up the container with Python 3.10 installed.
2 FROM python:3.10-slim (last pushed 3 weeks ago)
3
4 WORKDIR /app
5 COPY requirements.txt .
6 RUN pip install --no-cache-dir -r requirements.txt
7
8 COPY . .
9
10 # This tells Docker to listen on port 80 at runtime. Port 80 is the standard port for HTTP.
11 EXPOSE 80
12
13 # This command creates a .streamlit directory in the home directory of the container.
14 RUN mkdir ~/.streamlit
15
16 # This copies your Streamlit configuration file into the .streamlit directory you just created.
17 RUN cp config.toml ~/.streamlit/config.toml
18
19 # This sets the default command for the container to run app with Streamlit.
20 ENTRYPOINT ["streamlit", "run"]
21
22 # This command tells Streamlit to run your app when the container starts.
23 CMD ["app.py"]
```



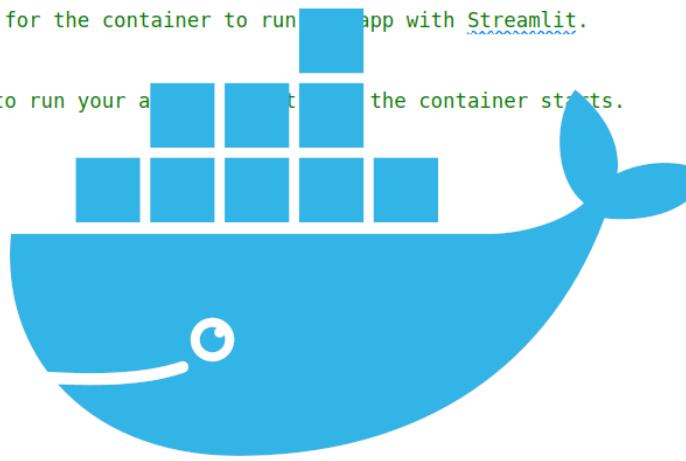
# Docker

A containerization platform

- packages application and its dependencies into a container
- ensures consistency across different environments
- easy to share and deploy
- widely used in industry, academia, research

```
compose.yml > ...
  ▷ Run All Services
1 services:
2   ▷ Run Service
3     resisto:
4       network_mode: bridge
5       working_dir: /app/app
6       ports:
7         - 8501:8501
8       image: resisto:latest
9       build: .
```

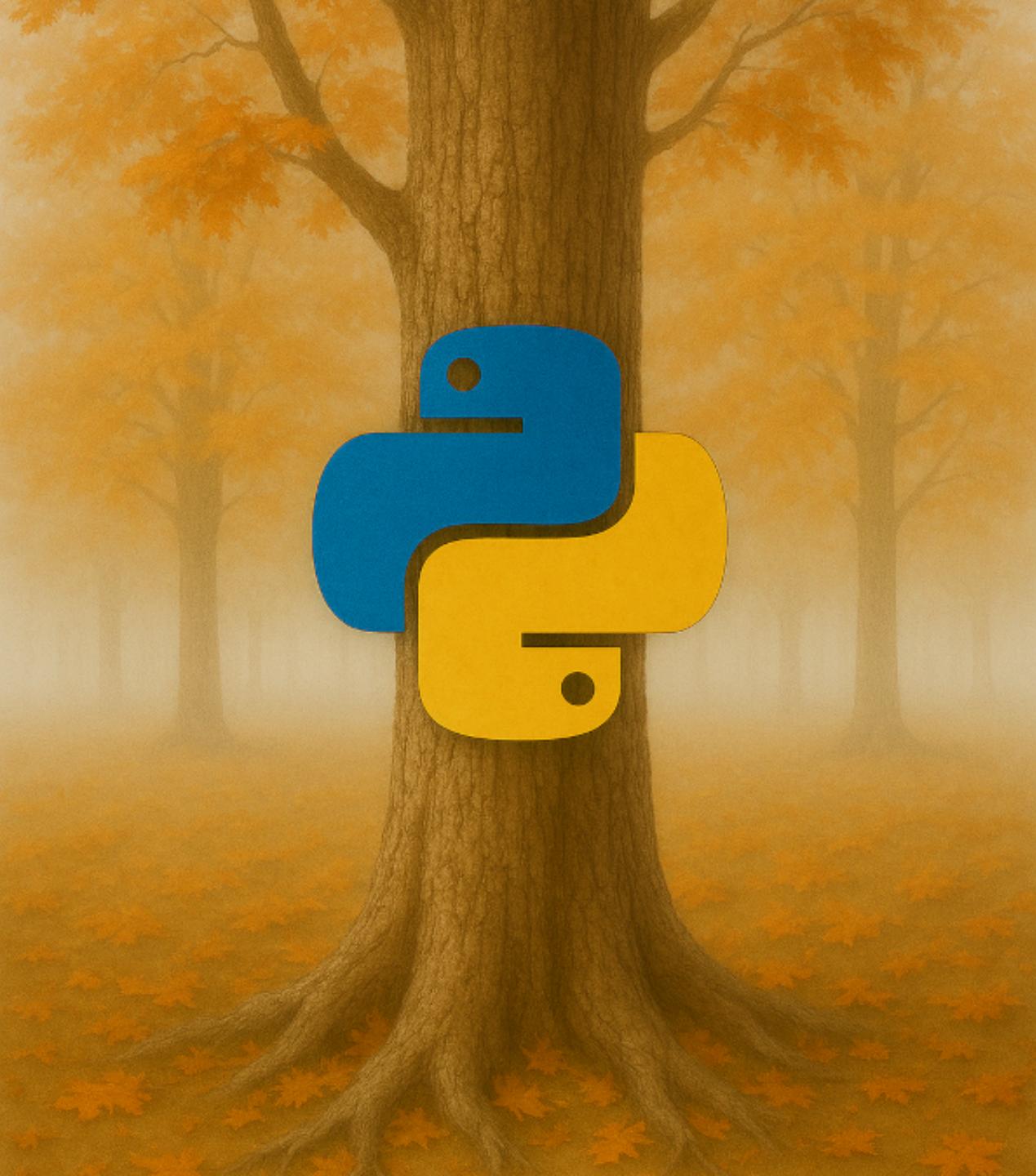
```
Dockerfile 1 ×
Dockerfile > ...
# This sets up the container with Python 3.10 installed.
1 FROM python:3.10-slim (last pushed 3 weeks ago)
2
3 WORKDIR /app
4 COPY requirements.txt .
5 RUN pip install --no-cache-dir -r requirements.txt
6
7 COPY ..
8
9 # This tells Docker to listen on port 80 at runtime. Port 80 is the standard port for HTTP.
10 EXPOSE 80
11
12 # This command creates a .streamlit directory in the home directory of the container.
13 RUN mkdir ~/.streamlit
14
15 # This copies your Streamlit configuration file into the .streamlit directory you just created.
16 RUN cp config.toml ~/.streamlit/config.toml
17
18 # This sets the default command for the container to run app with Streamlit.
19 ENTRYPOINT ["streamlit", "run"]
20
21 # This command tells Streamlit to run your app when the container starts.
22 CMD ["app.py"]
```



# Run dockerized app

docker compose up

- No Python install
- No dependency issues
- Works on Win / Mac / Linux
- Just clone repo with Dockerfile and docker-compose.yml
- First run = minutes, later = ms



# Summary

- Resistograph and tomograph are complementary tools for tree stem inspection
- A Python library was developed to simplify data merging and visualization
- GUI for Python is possible with Streamlit
- Installation can be made simple and repeatable with Docker