

**CollaboratiVessel.py**

```
1 import os
2 import sys
3
4 import logging
5 from typing import Annotated
6
7 import vtk
8
9 import csv
10 from datetime import datetime
11 import numpy as np
12
13 # QT is used for pop-ups
14 import qt
15
16 import slicer
17 from slicer.il8n import tr as _
18 from slicer.il8n import translate
19 from slicer.ScriptedLoadableModule import *
20 from slicer.util import VTKObservationMixin
21
22 from slicer import vtkMRMLScalarVolumeNode
23
24 from slicer.ScriptedLoadableModule import (
25     ScriptedLoadableModule,
26     ScriptedLoadableModuleWidget,
27     ScriptedLoadableModuleTest,
28 )
29
30 # Using the Lib folder to clean code
31 import importlib
32 _LIB_DIR = os.path.join(os.path.dirname(__file__), "Lib")
33 if _LIB_DIR not in sys.path:
```

```
34     sys.path.insert(0, _LIB_DIR)
35
36 if "CollaboratiVesselLogic" in sys.modules:
37     importlib.reload(sys.modules["CollaboratiVesselLogic"])
38
39 from CollaboratiVesselLogic import (
40     CollaboratiVesselLogic,
41     CollaboratiVesselParameterNode,
42 )
43
44 from helpers import rasToIjk
45
46
47 #
48 # CollaboratiVessel
49 # Widget files are preceded with dashes ----
50 # Logic files are preceded with equals ====
51 #
52
53 POINT_CLASSES = {
54     "vessel": {"label": "Vessel", "color": [1.0, 0.0, 0.0]},
55     "background": {"label": "Background", "color": [0.0, 0.0, 1.0]},
56     "outside": {"label": "Outside", "color": [0.0, 0.8, 0.0]},
57 }
58
59
60
61 class CollaboratiVessel(ScriptedLoadableModule):
62     """Uses ScriptedLoadableModule base class, available at:
63     https://github.com/Slicer/Slicer/blob/main/Base/Python/slicer/ScriptedLoadableModule.py
64     """
65
66     def __init__(self, parent):
67         ScriptedLoadableModule.__init__(self, parent)
68         self.parent.title = _("CollaboratiVessel")
69         # TODO: set categories (folders where the module shows up in the module selector)
```

```
70     self.parent.categories = [translate("qSlicerAbstractCoreModule", "Examples")]
71     self.parent.dependencies = [] # TODO: add here list of module names that this module requires
72     self.parent.contributors = ["Sean Nachtrab (UCL)"]
73
74     # TODO: update with short description of the module and a link to online module documentation
75     # _() function marks text as translatable to other languages
76     self.parent.helpText = _("Vessel segmentation assistant for low-contrast CECT data.")
77
78     # TODO: replace with organization, grant and thanks
79     self.parent.acknowledgementText = _("Developed for MSc thesis at UCLouvain.")
80
81     # Additional initialization step after application startup is complete
82     slicer.app.connect("startupCompleted()", registerSampleData)
83
84
85
86 #
87 # Register sample data sets in Sample Data module
88 #
89
90
91
92 def registerSampleData():
93     """
94     TODO: Fix the import from forge or use github
95     Add data sets to Sample Data module."""
96     # It is always recommended to provide sample data for users to make it easy to try the module,
97
98     import SampleData
99
100     iconsPath = os.path.join(os.path.dirname(__file__), "Resources/Icons")
101
102     # To ensure that the source code repository remains small (can be downloaded and installed quickly)
103     # it is recommended to store data sets that are larger than a few MB in a Github release.
104
105     # CollaboratiVessel1
```

```
106     SampleData.SampleDataLogic.registerCustomSampleDataSource(
107         # Category and sample name displayed in Sample Data module
108         category="CollaboratiVessel",
109         sampleName="SAMPLE SLICES CA-NM-L_x900_y900_z957",
110         # Thumbnail should have size of approximately 260x280 pixels and stored in Resources/Icons folder.
111         # It can be created by Screen Capture module, "Capture all views" option enabled, "Number of images" set to
"Single".
112         thumbnailFileName=os.path.join(iconsPath, "CollaboratiVessel1.png"),
113         # Download URL and target file name
114         # 8e58da8087bdd6453d48bfb08bed8bb3263d3806
115         uris="https://forge.uclouvain.be/SeanNachtrab/msc-thesis-sean-nachtrab/-/raw/main/code/example_data/
SAMPLE%20SLICES%20CA-NM-L_x900_y900_z957.nrrd",
116         fileNames="SAMPLE SLICES CA-NM-L_x900_y900_z957",
117         # Checksum to ensure file integrity. Can be computed by this command:
118         # import hashlib; print(hashlib.sha256(open(filename, "rb").read()).hexdigest())
119         # checksums="SHA256:998cb522173839c78657f4bc0ea907cea09fd04e44601f17c82ea27927937b95",
120         # This node name will be used when the data set is loaded
121         nodeNames="SAMPLE SLICES CA-NM-L_x900_y900_z957",
122         loadFileType='Volume Sequence'
123     )
124
125
126
127
128 #
129 # CollaboratiVesselWidget
130 # Based on the reference implementation
131 # https://www.slicer.org/wiki/Documentation/4.1/Developers/Modules#Scripted_Modules
132
133 # -----
134 # Widget elements use dash to identify them
135 # -----
136
137
138 class CollaboratiVesselWidget(ScriptedLoadableModuleWidget, VTKObservationMixin):
139     """Uses ScriptedLoadableModuleWidget base class, available at:
```

```
140 https://github.com/Slicer/Slicer/blob/main/Base/Python/slicer/ScriptedLoadableModule.py
141 """
142
143 def __init__(self, parent=None) -> None:
144     """Called when the user opens the module the first time and the widget is initialized."""
145     ScriptedLoadableModuleWidget.__init__(self, parent)
146     VTKObservationMixin.__init__(self) # needed for parameter node observation
147     self.logic = None
148     self._parameterNode = None
149     self._parameterNodeGuiTag = None
150
151     self._fiducialNodes = {}
152     self._fiducialObservers = {}
153     self._activePlacementClass = None
154     self._evalChunks = []
155     self._lastParams = {}
156
157
158 def setup(self) -> None:
159     """Called when the user opens the module the first time and the widget is initialized."""
160     ScriptedLoadableModuleWidget.setup(self)
161
162     # Load widget from .ui file (created by Qt Designer).
163     # Additional widgets can be instantiated manually and added to self.layout.
164     uiWidget = slicer.util.loadUI(self.resourcePath("UI/CollaboratiVessel.ui"))
165     self.layout.addWidget(uiWidget)
166     self.ui = slicer.util.childWidgetVariables(uiWidget)
167
168     # Set scene in MRML widgets. Make sure that in Qt designer the top-level qMRMLWidget's
169     # "mrmlSceneChanged(vtkMRMLScene*)" signal in is connected to each MRML widget's.
170     # "setMRMLScene(vtkMRMLScene*)" slot.
171     uiWidget.setMRMLScene(slicer.mrmlScene)
172
173     # Create logic class. Logic implements all computations that should be possible to run
174     # in batch mode, without a graphical user interface.
175     self.logic = CollaboratiVesselLogic()
```

```
176
177 # Create manager for markups
178 self._createFiducialNodes()
179
180 # Connections
181
182 # These connections ensure that we update parameter node when scene is closed
183 self.addObserver(slicer.mrmlScene, slicer.mrmlScene.StartCloseEvent, self.onSceneStartClose)
184 self.addObserver(slicer.mrmlScene, slicer.mrmlScene.EndCloseEvent, self.onSceneEndClose)
185
186
187 # Buttons
188 for cn in POINT_CLASSES:
189     self._buttonForClass("add", cn).connect(
190         "clicked(bool)", lambda _checked, c=cn: self.onStartPlacement(c))
191     self._buttonForClass("show", cn).connect(
192         "clicked(bool)", lambda _checked, c=cn: self.onShowPoint(c))
193     self._buttonForClass("delete", cn).connect(
194         "clicked(bool)", lambda _checked, c=cn: self.onDeletePoint(c))
195
196 self.ui.exportPointsButton.connect("clicked(bool)", self.onExportPoints)
197 self.ui.importPointsButton.connect("clicked(bool)", self.onImportPoints)
198
199 self.ui.runPipelineButton.connect("clicked(bool)", self.onRunPipeline)
200 self.ui.addEvalChunkButton.connect("clicked(bool)", self.onAddEvalChunk)
201 self.ui.clearEvalChunksButton.connect("clicked(bool)", self.onClearEvalChunks)
202 self.ui.computeMetricsButton.connect("clicked(bool)", self.onComputeMetrics)
203 self.ui.export2dButton.connect("clicked(bool)", self.onExport2D)
204 self.ui.loadExampleDataButton.connect("clicked(bool)", self.onLoadExampleData)
205
206 # Make sure parameter node is initialized (needed for module reload)
207 self.initializeParameterNode()
208
209
210 def onLoadExampleData(self):
211     """Sample data download
```

```
212     TODO: Hard coded name, revise"""
213     import SampleData
214     registerSampleData()
215     with slicer.util.tryWithErrorDisplay("Could not load sample data", waitCursor=True):
216         SampleData.SampleDataLogic().downloadFromSource(
217             SampleData.SampleDataLogic().sourceForSampleName("SLICES CA-NM-L_x+900_y+900_z+957"))
218
219
220 # Enable enlarging to other classes
221 def _buttonForClass(self, action, className):
222     return getattr(self.ui, f"{action}{className.capitalize()}PointButton")
223
224 def cleanup(self) -> None:
225     """Called when the application closes and the module widget is destroyed."""
226     self.removeObservers()
227     self._removeFiducialObservers()
228
229 # Update boxes
230 def _refreshAllComboBoxes(self):
231     for cn in POINT_CLASSES:
232         self._refreshComboBox(cn)
233
234 def enter(self) -> None:
235     """Called each time the user opens this module."""
236     # Make sure parameter node exists and observed
237     self.initializeParameterNode()
238     self._refreshAllComboBoxes()
239
240 def exit(self) -> None:
241     """Called each time the user opens a different module."""
242     # Do not react to parameter node changes (GUI will be updated when the user enters into the module)
243     if self._parameterNode:
244         self._parameterNode.disconnectGui(self._parameterNodeGuiTag)
245         self._parameterNodeGuiTag = None
246         # self.removeObserver(self._parameterNode, vtk.vtkCommand.ModifiedEvent, self._checkCanApply)
247
```

```
248 def onSceneStartClose(self, caller, event) -> None:
249     """Called just before the scene is closed."""
250     # Parameter node will be reset, do not use it anymore
251     self.setParameterNode(None)
252
253     self._removeFiducialObservers()
254     self._fiducialNodes.clear()
255
256 def onSceneEndClose(self, caller, event) -> None:
257     """Called just after the scene is closed."""
258     # If this module is shown while the scene is closed then recreate a new parameter node immediately
259     if self.parent.isEntered:
260         self.initializeParameterNode()
261         self._createFiducialNodes()
262         self._refreshAllComboBoxes()
263
264 def initializeParameterNode(self) -> None:
265     """Ensure parameter node exists and observed."""
266     # Parameter node stores all user choices in parameter values, node selections, etc.
267     # so that when the scene is saved and reloaded, these settings are restored.
268
269     self.setParameterNode(self.logic.getParameterNode())
270
271     # Select default input nodes if nothing is selected yet to save a few clicks for the user
272     if not self._parameterNode.inputVolume:
273         firstVolumeNode = slicer.mrmlScene.GetFirstNodeByClass("vtkMRMLScalarVolumeNode")
274         if firstVolumeNode:
275             self._parameterNode.inputVolume = firstVolumeNode
276
277 def setParameterNode(self, inputParameterNode: CollaboratiVesselParameterNode | None) -> None:
278     """
279     Set and observe parameter node.
280     Observation is needed because when the parameter node is changed then the GUI must be updated immediately.
281     """
282
283     if self._parameterNode:
```

```
284         self._parameterNode.disconnectGui(self._parameterNodeGuiTag)
285         # self.removeObserver(self._parameterNode, vtk.vtkCommand.ModifiedEvent, self._checkCanApply)
286     self._parameterNode = inputParameterNode
287     if self._parameterNode:
288         # Note: in the .ui file, a Qt dynamic property called "SlicerParameterName" is set on each
289         # ui element that needs connection.
290         self._parameterNodeGuiTag = self._parameterNode.connectGui(self.ui)
291         # self.addObserver(self._parameterNode, vtk.vtkCommand.ModifiedEvent, self._checkCanApply)
292         # self._checkCanApply()
293
294     # -----
295     # End of setup
296     # -----
297
298
299
300     # -----
301     # Running the pipeline
302     # -----
303
304     def onRunPipeline(self):
305         vol = self._getVolumeOrError()
306         if not vol:
307             return
308
309         vesselIJKs = self._seedIJKs(vol, "vessel")
310         if not vesselIJKs:
311             slicer.util.errorDisplay("Place at least one vessel point first.")
312             return
313         bgIJKs = self._seedIJKs(vol, "background")
314
315         # Can contain nothing if no outside
316         outsideIJKs = self._seedIJKs(vol, "outside")
317
318         params = {
319             "vessel_size": self.ui.avgVesselSizeSpinBox.value,
```

```
320         "vessel_std":                self.ui.vesselSizeDeviationSpinBox.value,
321         "structure_strength":         self.ui.structureStrengthSpinBox.value,
322         "frangi_tiles_per_axis":      self.ui.frangiTilesPerAxisSpinBox.value,
323         # "intensity_likelihood":      self.ui.intensityLikelihoodCheckBox.checked,
324         # "vessel_prior":              self.ui.vesselPriorSpinBox.value,
325         "ridge_threshold":            self.ui.ridgeThresholdSpinBox.value,
326         "ridge_iterations":           self.ui.ridgeIterationsSpinBox.value,
327         # "auto_seed":                 self.ui.autoSeedCheckBox.checked,
328         "auto_seed_rounds":           self.ui.autoSeedRoundsSpinBox.value,
329         "auto_seed_max":              self.ui.autoSeedMaxSpinBox.value,
330         # "auto_seed_intensity_sigma": self.ui.autoSeedIntensitySigmaSpinBox.value,
331         # "component_min_prob":        self.ui.componentMinProbSpinBox.value,
332     }
333
334     # Remember the params for onComputeMetrics's CSV export.
335     self._lastParams = dict(params)
336
337     import time
338     t0 = time.time()
339
340     self._setResults("Running pipeline...")
341     slicer.app.processEvents()
342
343     with slicer.util.tryWithErrorDisplay("Pipeline failed.", waitCursor=True):
344         state = self.logic.runPipeline(vol, vesselIJKs, bgIJKs, outsideIJKs, params)
345
346     elapsed = time.time() - t0
347     self._publishOutputs(state, vol, elapsed=elapsed)
348     self._refreshEvalUI()
349
350     # Place outputs in a user friendly format
351     def _publishOutputs(self, state, vol, elapsed=None):
352         # self.logic.pushArray("VSA_Probability", state["probMap"], vol)
353         self.logic.pushArray("Frangi", state["frangi"], vol)
354         self.logic.pushArray("p_intensity", state["p_intensity"], vol)
355         self.logic.pushArray("probMap", state["probMap"], vol)
```

```
356 self.logic.pushArray("soft_mask", state["soft_mask"], vol)
357 self.logic.pushArray("binMask", state["binMask"], vol)
358
359 self.logic.createSegmentationFromMask(
360     state["binMask"], vol, segName="VSA_Vessels")
361
362 spacing = vol.GetSpacing()
363 voxelVol = float(np.prod(spacing))
364 nVox = int(state["binMask"].sum())
365 report = ["<b>Pipeline complete</b>"]
366
367 if elapsed is not None:
368     report.append(f"Time: {elapsed:.1f}s")
369     report.append(f"Vessel voxels: {nVox:,} ({nVox * voxelVol:.3f} mm³)")
370
371 # Seed-vs-prediction agreement: sanity check that doesn't need GT.
372 # A vessel point is 'correct' if the prediction is True at that
373 # voxel; a background point if it's False.
374 agree = self._seedAgreement(vol, state["binMask"])
375 if agree:
376     report.append("")
377     report.append("<b>Seed agreement (vs prediction):</b>")
378     for cn, st in agree.items():
379         if st["n"] == 0:
380             continue
381         pct = (100.0 * st["accuracy"]) if not np.isnan(st["accuracy"]) else float("nan")
382         report.append(
383             f" • {cn}: {st['correct']}/{st['n']} "
384             f"correctly classified ({pct:.1f}%)"
385         )
386
387 report.append("")
388 report.append("<b>Stages:</b>")
389 report.extend(f" • {line}" for line in state["log"])
390 self._setResults("<br>".join(report))
391
```

```
392 def _seedAgreement(self, vol, mask):
393     """For each seed class, how many points sit on the 'right side' of
394     the prediction. Vessel points should be inside (mask=True),
395     background points should be outside (mask=False). 'outside' class
396     is intentionally skipped (not used by the pipeline)."""
397     if mask is None or vol is None:
398         return None
399     seedsByClass = self._collectSeedPoints()
400     out = {}
401     for cn in ("vessel", "background"):
402         seedRas = seedsByClass.get(cn, [])
403         n = len(seedRas)
404         correct = 0
405         for ras in seedRas:
406             ijk = self.logic.rasToIjk(vol, ras, clamp=True)
407             if ijk is None:
408                 continue
409             i, j, k = ijk
410             if (0 <= k < mask.shape[0] and
411                 0 <= j < mask.shape[1] and
412                 0 <= i < mask.shape[2]):
413                 inMask = bool(mask[k, j, i])
414                 if (cn == "vessel" and inMask) or \
415                     (cn == "background" and not inMask):
416                     correct += 1
417         out[cn] = {
418             "n": n,
419             "correct": correct,
420             "accuracy": (correct / n) if n > 0 else float("nan"),
421         }
422     return out
423
424
425
426 # -----
427 # Evaluation
```

```
428 # -----
429
430 def _refreshEvalUI(self):
431     n = len(self._evalChunks)
432     self.ui.evalChunksStatusLabel.setText(f"Chunks loaded: {n} / 3")
433     self.ui.addEvalChunkButton.setEnabled(n < 3)
434     self.ui.computeMetricsButton.setEnabled(
435         n > 0 and self.logic.state.get("binMask") is not None
436     )
437
438 @staticmethod
439 def _loadAnnotationVolume(path):
440     if not path or not os.path.exists(path):
441         return None
442     if os.path.isfile(path):
443         try:
444             return slicer.util.loadVolume(path)
445         except Exception as e:
446             print(e)
447             return None
448     for fname in sorted(os.listdir(path)):
449         full = os.path.join(path, fname)
450         if not os.path.isfile(full):
451             continue
452         try:
453             node = slicer.util.loadVolume(full)
454             if node is not None:
455                 return node
456         except Exception as e:
457             print(e)
458             continue
459     return None
460
461 def onAddEvalChunk(self):
462     if len(self._evalChunks) >= 3:
463         return
```

```
464     path = qt.QFileDialog.getExistingDirectory(
465         None, "Select annotation folder (DICOM or with single volume file)"
466     )
467     if not path:
468         path = qt.QFileDialog.getOpenFileName(
469             None, "Select annotation file",
470             "", "Volumes (*.nrrd *.nii *.nii.gz *.dcm);;All files (*)"
471         )
472         if not path:
473             return
474
475     with slicer.util.tryWithErrorDisplay("Annotation load failed.",
476                                         waitCursor=True):
477         node = self._loadAnnotationVolume(path)
478
479     if node is None:
480         slicer.util.errorDisplay(
481             f"Could not load a volume from:\n{path}\n"
482             "Supported: DICOM folder, NRRD, NIfTI."
483         )
484         return
485
486     node.SetName(f"VSA_Eval_{len(self._evalChunks) + 1}")
487     self._evalChunks.append(node)
488     self._refreshEvalUI()
489     self._appendToResults(
490         f"<br>Loaded evaluation chunk #{len(self._evalChunks)}: "
491         f"{node.GetName()} from {path}"
492     )
493
494     def onClearEvalChunks(self):
495         for node in self._evalChunks:
496             try:
497                 slicer.mrmlScene.RemoveNode(node)
498             except Exception as e:
499                 print(e)
```

```
500         pass
501
502         self._evalChunks = []
503         self._refreshEvalUI()
504         self._appendToResults("<br>Cleared all evaluation chunks.")
505
506
507     # Show the metrics to the user
508     # Formatted for readability using HTML tags
509     # TODO: Could output this to an JSON
510     def onComputeMetrics(self):
511         vol = self._getVolumeOrError()
512         if not vol:
513             return
514         mask = self.logic.state.get("binMask")
515         if mask is None:
516             slicer.util.errorDisplay("Run the pipeline first.")
517             return
518         if not self._evalChunks:
519             slicer.util.errorDisplay("Add at least one evaluation chunk.")
520             return
521
522         # Optional grey-value threshold for the baseline comparison.
523         # Empty / unparseable input disables the threshold baseline.
524         threshold = None
525         thr_text = self.ui.thresholdLineEdit.text.strip()
526         if thr_text:
527             try:
528                 threshold = float(thr_text)
529             except ValueError:
530                 slicer.util.errorDisplay(
531                     f"Threshold must be a number (got: {thr_text!r}).")
532             return
533
534         with slicer.util.tryWithErrorDisplay("Metric computation failed.",
535                                             waitCursor=True):
```

```

536         results = self.logic.evaluateAgainstChunks(
537             self._evalChunks, vol, mask, threshold=threshold
538         )
539
540     IND = "&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;"
541     lines = ["<b>Evaluation against ground-truth chunks</b>",
542             "<i>(metrics on GT&gt;0 bbox + 5-voxel margin)</i>"]
543     for r in results["per_chunk"]:
544         if "note" in r:
545             lines.append(f"    • {r['name']}: {r['note']}")
546             continue
547         x0, y0, z0, x1, y1, z1 = r["source_ijk"]
548         lines.append(
549             f"    • {r['name']}: "
550             f"<b>Dice={r['dice']:.3f}</b>, cLDice={r['cldice']:.3f}, "
551             f"IoU={r['iou']:.3f}, "
552             f"P={r['precision']:.3f}, R={r['recall']:.3f}, "
553             f"vol(pred/gt)={r['volume_ratio']:.2f}<br>"
554             f"{IND}GT={r['gt_voxels']:}, vox, pred={r['pred_voxels']:}, vox, "
555             f"shape={r['shape']], "
556             f"src IJK=[{x0}:{x1}, {y0}:{y1}, {z0}:{z1}]"
557         )
558         pc, gc = r["pred_centerline"], r["gt_centerline"]
559         lines.append(
560             f"{IND}<i>Centerlines:</i> "
561             f"pred={pc['length_mm']:.1f}mm ({pc['n_branches']} br, "
562             f"avg {pc['mean_branch_length_mm']:.1f}mm), "
563             f"gt={gc['length_mm']:.1f}mm ({gc['n_branches']} br, "
564             f"avg {gc['mean_branch_length_mm']:.1f}mm)"
565         )
566         if "dice_thr_vs_gt" in r:
567             tg = r["thr_vs_gt"]
568             pt = r["pred_vs_thr"]
569             tc = r["thr_centerline"]
570             lines.append(
571                 f"{IND}<i>Threshold ({r['threshold value']:g}) vs GT:</i> "

```

```
572         f"Dice={r['dice_thr_vs_gt']:.3f}, "
573         f"clDice={r['cldice_thr_vs_gt']:.3f}, "
574         f"IoU={tg['iou']:.3f}, "
575         f"P={tg['precision']:.3f}, R={tg['recall']:.3f}, "
576         f"vol(thr/gt)={tg['volume_ratio']:.2f}, "
577         f"thr={r['thr_voxels']:,} vox<br>"
578         f"{IND}<i>Pred vs Threshold:</i> "
579         f"Dice={r['dice_pred_vs_thr']:.3f}, "
580         f"clDice={r['cldice_pred_vs_thr']:.3f}, "
581         f"IoU={pt['iou']:.3f}, "
582         f"vol(pred/thr)={pt['volume_ratio']:.2f}<br>"
583         f"{IND}<i>Threshold centerline:</i> "
584         f"thr={tc['length_mm']:.1f}mm ({tc['n_branches']} br, "
585         f"avg {tc['mean_branch_length_mm']:.1f}mm)"
586     )
587     if results["n"] > 0:
588         lines.append(
589             f"<b>Mean – Pred vs GT:</b> "
590             f"Dice={results['mean_dice']:.3f}, "
591             f"clDice={results['mean_cldice']:.3f}, "
592             f"IoU={results['mean_iou']:.3f}, "
593             f"P={results['mean_precision']:.3f}, "
594             f"R={results['mean_recall']:.3f}, "
595             f"vol={results['mean_volume_ratio']:.2f}<br>"
596             f"<b>Mean centerline length:</b> "
597             f"pred={results['mean_pred_length_mm']:.1f}mm, "
598             f"gt={results['mean_gt_length_mm']:.1f}mm"
599         )
600     if "mean_dice_thr_vs_gt" in results:
601         lo, hi = results.get("src_value_range", (float("nan"), float("nan")))
602         lines.append(
603             f"<i>Threshold diag:</i> threshold={results['threshold_value']:g}; "
604             f"total thresh voxels in volume: {results['total_thr_voxels']:,}; "
605             f"source value range: [{lo:.1f}, {hi:.1f}]"
606         )
607     lines.append(
```

```
608         f"<b>Mean – Threshold vs GT:</b> "
609         f"Dice={results['mean_dice_thr_vs_gt']:.3f}, "
610         f"clDice={results['mean_cldice_thr_vs_gt']:.3f}, "
611         f"IoU={results['mean_iou_thr_vs_gt']:.3f}, "
612         f"P={results['mean_precision_thr_vs_gt']:.3f}, "
613         f"R={results['mean_recall_thr_vs_gt']:.3f}, "
614         f"vol={results['mean_volume_ratio_thr_vs_gt']:.2f}<br>"
615         f"<b>Mean – Pred vs Threshold:</b> "
616         f"Dice={results['mean_dice_pred_vs_thr']:.3f}, "
617         f"clDice={results['mean_cldice_pred_vs_thr']:.3f}, "
618         f"IoU={results['mean_iou_pred_vs_thr']:.3f}<br>"
619         f"<b>Mean threshold centerline:</b> "
620         f"thr={results['mean_thr_length_mm']:.1f}mm"
621     )
622     self._appendToResults("<br>" + "<br>".join(lines))
623
624     # Structured CSV (one row per evaluable chunk). Renders inside
625     # <pre> so column alignment / commas are preserved for copy-paste.
626     csvText = self._buildMetricsCsv(results)
627     if csvText:
628         self._appendToResults(
629             "<br><b>CSV:</b><pre style='font-family:monospace'>"
630             + csvText + "</pre>"
631         )
632
633     # Column layout for the metrics CSV. Order matches the row builder
634     # below; change them together.
635     _CSV_PARAM_KEYS = (
636         "vessel_size", "vessel_std", # "denoise_kernel",
637         "frangi_tiles_per_axis",
638         # "intensity_likelihood", "vessel_prior",
639         "ridge_threshold", "ridge_iterations",
640         # "auto_seed",
641         "auto_seed_rounds", "auto_seed_max",
642         # "auto_seed_intensity_sigma", "component_min_prob",
643     )
```

```
644
645 @staticmethod
646 def _fmt(v, nd=4):
647     try:
648         return f"{float(v):.{nd}f}"
649     except (TypeError, ValueError):
650         return ""
651
652 def _buildMetricsCsv(self, results):
653     import io, csv as _csv
654
655     thresholded = any("dice_thr_vs_gt" in r for r in results["per_chunk"])
656
657     # Seed-vs-prediction agreement is volume-level, so the same four
658     # numbers are repeated across chunk rows. Aggregation tools can
659     # collapse them.
660     vol = self.getActiveVolume()
661     mask = self.logic.state.get("binMask")
662     agree = self._seedAgreement(vol, mask) or {}
663     agreeCols = (
664         int(agree.get("vessel", {}).get("correct", 0)),
665         int(agree.get("vessel", {}).get("n", 0)),
666         int(agree.get("background", {}).get("correct", 0)),
667         int(agree.get("background", {}).get("n", 0)),
668     )
669
670     header = ["chunk"] + list(self._CSV_PARAM_KEYS) + [
671         "median_intensity", "std_intensity",
672         "vessel_seeds_correct", "vessel_seeds_total",
673         "bg_seeds_correct", "bg_seeds_total",
674         "pred_gt_dice", "pred_gt_cldice", "pred_gt_iou",
675         "pred_gt_precision", "pred_gt_recall", "pred_gt_vol",
676     ]
677     if thresholded:
678         header += [
679             "thr_gt_dice", "thr_gt_cldice", "thr_gt_iou",
```

```
680         "thr_gt_precision", "thr_gt_recall", "thr_gt_vol",
681         "pred_thr_dice", "pred_thr_cldice", "pred_thr_iou",
682         "pred_thr_precision", "pred_thr_recall", "pred_thr_vol",
683     ]
684     header += ["cl_len_gt_mm", "cl_len_pred_mm", "cl_match_pred_mm"]
685     if thresholded:
686         header += ["cl_len_thr_mm", "cl_match_thr_mm"]
687
688     buf = io.StringIO()
689     w = _csv.writer(buf)
690     w.writerow(header)
691
692     fmt = self._fmt
693     for r in results["per_chunk"]:
694         if "note" in r:
695             continue
696         row = [r["name"]]
697         for k in self._CSV_PARAM_KEYS:
698             row.append(self._lastParams.get(k, ""))
699         row += [fmt(r["median_intensity"], 2), fmt(r["std_intensity"], 2)]
700         row += list(agreeCols)
701         row += [
702             fmt(r["dice"]), fmt(r["cldice"]), fmt(r["iou"]),
703             fmt(r["precision"]), fmt(r["recall"]),
704             fmt(r["volume_ratio"], 3),
705         ]
706         if thresholded:
707             tg, pt = r["thr_vs_gt"], r["pred_vs_thr"]
708             row += [
709                 fmt(r["dice_thr_vs_gt"]), fmt(r["cldice_thr_vs_gt"]),
710                 fmt(tg["iou"]), fmt(tg["precision"]),
711                 fmt(tg["recall"]), fmt(tg["volume_ratio"], 3),
712                 fmt(r["dice_pred_vs_thr"]), fmt(r["cldice_pred_vs_thr"]),
713                 fmt(pt["iou"]), fmt(pt["precision"]),
714                 fmt(pt["recall"]), fmt(pt["volume_ratio"], 3),
715             ]
```

```
716         gc = r["gt_centerline"]
717         pc = r["pred_centerline"]
718         row += [fmt(gc["length_mm"], 2),
719                fmt(pc["length_mm"], 2),
720                fmt(pc.get("matching_length_mm"), 2)]
721         if thresholded:
722             tc = r["thr_centerline"]
723             row += [fmt(tc["length_mm"], 2),
724                    fmt(tc.get("matching_length_mm"), 2)]
725         w.writerow(row)
726
727     return buf.getvalue().rstrip("\r\n")
728
729
730
731 # -----
732 # Export 2D binary slices
733 # This format is universal but lags 3D Slicer if re-imported
734 # -----
735
736 def onExport2D(self):
737     mask = self.logic.state.get("binMask")
738     if mask is None:
739         slicer.util.errorDisplay("Run the pipeline first.")
740         return
741
742     outDir = qt.QFileDialog.getExistingDirectory(
743         None, "Select directory for 2D slice export")
744     if not outDir:
745         return
746
747     vol = self.getActiveVolume()
748     prefix = "vessel"
749     if vol is not None:
750         safe = "".join(c if c.isalnum() or c in "-_." else "_"
751                        for c in vol.GetName())
```

```
752         prefix = f"{safe}_vessel"
753
754     with slicer.util.tryWithErrorDisplay("Slice export failed.", waitCursor=True):
755         n = self.logic.exportBinarySlices(mask, outDir, prefix=prefix, axis=0)
756
757     self._appendToResults(f"<br><b>Exported {n} slices to:</b><br>{outDir}")
758     slicer.util.infoDisplay(f"Exported {n} slices to:\n{outDir}")
759
760
761
762     # -----
763     # Show results
764     # -----
765
766     def _setResults(self, text):
767         self.ui.resultsTextBrowser.setHtml(text)
768
769     def _appendToResults(self, text):
770         cur = self.ui.resultsTextBrowser.toHtml()
771         self.ui.resultsTextBrowser.setHtml(cur + text)
772
773
774
775     # -----
776     # Fiducial node management
777     # -----
778
779     def _createFiducialNodes(self):
780         for className, info in POINT_CLASSES.items():
781             nodeName = f"VSA_{info['label']}"
782             existing = slicer.mrmlScene.GetNodesByName(nodeName)
783             if existing.GetNumberOfItems() > 0:
784                 node = existing.GetItemAsObject(0)
785             else:
786                 node = slicer.mrmlScene.AddNewNodeByClass(
787                     "vtkMRMLMarkupsFiducialNode", nodeName)
```

```
788         d = node.GetDisplayNode()
789         if d:
790             d.SetSelectedColor(*info["color"])
791             d.SetColor(*info["color"])
792             d.SetGlyphScale(2.0)
793             d.SetTextScale(2.0)
794         self._fiducialNodes[className] = node
795         tag = node.AddObserver(
796             slicer.vtkMRMLMarkupsNode.PointPositionDefinedEvent,
797             lambda caller, event, cn=className: self._refreshComboBox(cn))
798         self._fiducialObservers[className] = tag
799
800     def _removeFiducialObservers(self):
801         for className, node in self._fiducialNodes.items():
802             tag = self._fiducialObservers.get(className)
803             if tag is not None:
804                 node.RemoveObserver(tag)
805         self._fiducialObservers.clear()
806
807     def _comboBoxForClass(self, className):
808         return getattr(self.ui, f"{className}PointsComboBox")
809
810     def _refreshComboBox(self, className):
811         cb = self._comboBoxForClass(className)
812         if cb is None:
813             return
814         node = self._fiducialNodes[className]
815         cb.blockSignals(True)
816         cb.clear()
817         for i in range(node.GetNumberOfControlPoints()):
818             ras = [0.0, 0.0, 0.0]
819             node.GetNthControlPointPosition(i, ras)
820             cb.addItem(f"Pt {i}: [{ras[0]:.1f}, {ras[1]:.1f}, {ras[2]:.1f}]", i)
821         cb.blockSignals(False)
822         if cb.count > 0:
823             cb.setCurrentIndex(cb.count - 1)
```

```
824
825 # Non internal
826 def onStartPlacement(self, className):
827     node = self._fiducialNodes.get(className)
828     if node is None:
829         return
830     self._activePlacementClass = className
831     sel = slicer.mrmlScene.GetNodeByID("vtkMRMLSelectionNodeSingleton")
832     sel.SetActivePlaceNodeID(node.GetID())
833     ix = slicer.mrmlScene.GetNodeByID("vtkMRMLInteractionNodeSingleton")
834     ix.SetPlaceModePersistence(1)
835     ix.SetCurrentInteractionMode(ix.Place)
836
837 def onShowPoint(self, className):
838     cb = self._comboBoxForClass(className)
839     idx = cb.currentIndex
840     if idx < 0:
841         return
842     node = self._fiducialNodes[className]
843     if idx >= node.GetNumberOfControlPoints():
844         return
845     ras = [0.0, 0.0, 0.0]
846     node.GetNthControlPointPosition(idx, ras)
847     slicer.modules.markups.logic().JumpSlicesToLocation(*ras, True)
848
849 def onDeletePoint(self, className):
850     cb = self._comboBoxForClass(className)
851     idx = cb.currentIndex
852     if idx < 0:
853         return
854     node = self._fiducialNodes[className]
855     if idx < node.GetNumberOfControlPoints():
856         node.RemoveNthControlPoint(idx)
857         self._refreshComboBox(className)
858
859
```

```
860
861 # -----
862 # Widget specific helpers
863 # -----
864
865 # Bound to the widget class. Callable
866 @staticmethod
867 def getActiveVolume():
868     layoutManager = slicer.app.layoutManager()
869     if layoutManager is not None:
870         for sliceName in layoutManager.sliceViewNames():
871             w = layoutManager.sliceWidget(sliceName)
872             if w is None:
873                 continue
874             cn = w.sliceLogic().GetSliceCompositeNode()
875             vid = cn.GetBackgroundVolumeID()
876             if vid:
877                 n = slicer.mrmlScene.GetNodeByID(vid)
878                 if n is not None:
879                     return n
880         for cls in ["vtkMRMLScalarVolumeNode", "vtkMRMLVectorVolumeNode"]:
881             n = slicer.mrmlScene.GetFirstNodeByClass(cls)
882             if n is not None:
883                 return n
884     return None
885
886 def _getVolumeOrError(self):
887     vol = self.getActiveVolume()
888     if not vol:
889         slicer.util.errorDisplay("No volume found. Load and view your data first.")
890         return None
891     if vol.GetImageData() is None:
892         slicer.util.errorDisplay("Active volume has no image data.")
893         return None
894     return vol
895
```

```
896 def _collectSeedPoints(self):
897     out = {}
898     for cn in POINT_CLASSES:
899         node = self._fiducialNodes[cn]
900         pts = []
901         for i in range(node.GetNumberOfControlPoints()):
902             ras = [0.0, 0.0, 0.0]
903             node.GetNthControlPointPosition(i, ras)
904             pts.append(ras)
905         out[cn] = pts
906     return out
907
908 def _seedIJKs(self, vol, className):
909     out = []
910     for ras in self._collectSeedPoints().get(className, []):
911         ijk = rasToIjk(vol, ras, clamp=True)
912         if ijk is not None:
913             out.append(ijk)
914     return out
915
916
917
918 # -----
919 # CSV import / export of points
920 # -----
921
922 def onExportPoints(self):
923     seedPoints = self._collectSeedPoints()
924     total = sum(len(p) for p in seedPoints.values())
925     if total == 0:
926         slicer.util.errorDisplay("No points to export.")
927         return
928
929     vol = self.getActiveVolume()
930     volName = vol.GetName() if vol else "unknown_volume"
931     safe = "".join(c if c.isalnum() or c in "-_." else "_" for c in volName)
```

```
932     ts = datetime.now().strftime("%Y%m%d_%H%M%S")
933     defaultName = f"{safe}_seeds_{ts}.csv"
934
935     path = qt.QFileDialog.getSaveFileName(
936         None, "Export Seed Points", defaultName, "CSV files (*.csv)")
937     if not path:
938         return
939
940     with open(path, "w", newline="") as f:
941         f.write("# VSA seed points export\n")
942         f.write(f"# volume: {volName}\n")
943         f.write(f"# date: {datetime.now().isoformat()}\n")
944         f.write(f"# total_points: {total}\n")
945         w = csv.writer(f)
946         w.writerow(["class", "index", "R", "A", "S"])
947         for cn, pts in seedPoints.items():
948             for i, ras in enumerate(pts):
949                 w.writerow([cn, i, ras[0], ras[1], ras[2]])
950     slicer.util.infoDisplay(f"Exported {total} points to:\n{path}")
951
952     def onImportPoints(self):
953         path = qt.QFileDialog.getOpenFileName(
954             None, "Import Seed Points", "", "CSV files (*.csv)")
955         if not path:
956             return
957         imported = {cn: [] for cn in POINT_CLASSES}
958         with open(path, "r") as f:
959             lines = [ln for ln in f if not ln.startswith("#")]
960         for row in csv.DictReader(lines):
961             cn = row["class"]
962             if cn not in POINT_CLASSES:
963                 continue
964             imported[cn].append(
965                 [float(row["R"]), float(row["A"]), float(row["S"])]])
966
967         for cn, rasList in imported.items():
```

```
968         node = self._fiducialNodes[cn]
969         node.RemoveAllControlPoints()
970         for ras in rasList:
971             node.AddControlPoint(*ras)
972         node.CreateDefaultDisplayNodes()
973         d = node.GetDisplayNode()
974         if d:
975             color = POINT_CLASSES[cn]["color"]
976             d.SetSelectedColor(*color)
977             d.SetColor(*color)
978             d.SetVisibility(True)
979             d.SetVisibility3D(True)
980             d.SetVisibility2D(True)
981         self._refreshComboBox(cn)
982
983     total = sum(len(p) for p in imported.values())
984     slicer.util.infoDisplay(f"Imported {total} points from:\n{path}")
985
986
987
988
989
990
991 #
992 # CollaboratiVesselTest
993 #
994
995
996 class CollaboratiVesselTest(ScriptedLoadableModuleTest):
997     """
998     This is the test case for your scripted module.
999     Uses ScriptedLoadableModuleTest base class, available at:
1000     https://github.com/Slicer/Slicer/blob/main/Base/Python/slicer/ScriptedLoadableModule.py
1001     """
1002
1003     def setUp(self):
```

```
1004         """Do whatever is needed to reset the state - typically a scene clear will be enough."""
1005         slicer.mrmlScene.Clear()
1006
1007     def runTest(self):
1008         """Run as few or as many tests as needed here."""
1009         self.setUp()
1010         self.test_CollaboratiVessel1()
1011
1012     def test_CollaboratiVessel1(self):
1013         """
1014         TODO: Testing not currently done
1015
1016         Ideally you should have several levels of tests. At the lowest level
1017         tests should exercise the functionality of the logic with different inputs
1018         (both valid and invalid). At higher levels your tests should emulate the
1019         way the user would interact with your code and confirm that it still works
1020         the way you intended.
1021         One of the most important features of the tests is that it should alert other
1022         developers when their changes will have an impact on the behavior of your
1023         module. For example, if a developer removes a feature that you depend on,
1024         your test should break so they know that the feature is needed.
1025         """
1026
1027         self.delayDisplay("Starting the test")
1028
1029         # Get/create input data
1030
1031         import SampleData
1032
1033         registerSampleData()
1034         inputVolume = SampleData.downloadSample("CollaboratiVessel1")
1035         self.delayDisplay("Loaded test data set")
1036
1037         inputScalarRange = inputVolume.GetImageData().GetScalarRange()
1038         self.assertEqual(inputScalarRange[0], 0)
1039         self.assertEqual(inputScalarRange[1], 695)
```

```
1040
1041     outputVolume = slicer.mrmlScene.AddNewNodeByClass("vtkMRMLScalarVolumeNode")
1042     threshold = 100
1043
1044     # Test the module logic
1045
1046     logic = CollaboratiVesselLogic()
1047
1048     # Test algorithm with non-inverted threshold
1049     logic.process(inputVolume, outputVolume, threshold, True)
1050     outputScalarRange = outputVolume.GetImageData().GetScalarRange()
1051     self.assertEqual(outputScalarRange[0], inputScalarRange[0])
1052     self.assertEqual(outputScalarRange[1], threshold)
1053
1054     # Test algorithm with inverted threshold
1055     logic.process(inputVolume, outputVolume, threshold, False)
1056     outputScalarRange = outputVolume.GetImageData().GetScalarRange()
1057     self.assertEqual(outputScalarRange[0], inputScalarRange[0])
1058     self.assertEqual(outputScalarRange[1], inputScalarRange[1])
1059
1060     self.delayDisplay("Test passed")
1061
```