Here's a comprehensive specification of all files in the project, with focus points for RS2 interaction and model adjustment:

**1. Core Files**

**core/calibrator.py**

* **Purpose**: Main calibration workflow controller
* **Key Functions**:
  + evaluate\_parameters(): Runs simulations and compares with experimental data
  + \_adjust\_for\_pressure(): Handles pressure-dependent parameter adjustments
* **RS2 Adjustments**:
  + Modify \_run\_simulation() to customize RS2 model updates
  + Change error metrics in \_calculate\_error()

**core/material\_model.py**

* **Purpose**: Material model definitions
* **Key Classes**:
  + NorSand: Default implementation
* **Customization**:
  + Add new models (e.g., MohrCoulomb, HardeningSoil)
  + Modify parameter bounds/ranges

**2. RS2 Interface Files**

**interfaces/rs2\_interface.py**

* **Critical RS2 Functions**:

python

Copy

def create\_test\_model(self, material, parameters, cell\_pressure, drainage):

# Create triaxial test model in RS2

self.model.MaterialProperties.AddMaterial().Behavior = MaterialType.NorSand

self.model.Stages[0].CellPressure = cell\_pressure

self.model.Stages[0].Drainage = DrainageType.Drained if drainage == "drained" else DrainageType.Undrained

def get\_stress\_strain(self):

# Extract results from RS2

results = self.model.GetResults()

return {

'strain': results.GetColumn("Axial Strain"),

'stress': results.GetColumn("Deviator Stress"),

'pore\_pressure': results.GetColumn("Pore Pressure")

}

* **Adjustment Points**:
  + Change stage configuration in create\_test\_model()
  + Modify result extraction in get\_stress\_strain()

**interfaces/data\_loader.py**

* **Purpose**: Load experimental data
* **Key Function**:

python

Copy

def load\_test(self, filepath, test\_type, cell\_pressure):

# Returns dict with keys: strain, stress, pore\_pressure

**3. Optimization Files**

**optimization/pressure\_optimizer.py**

* **Purpose**: Handles parameter optimization
* **Key Methods**:
  + optimize(): Runs hybrid DE + L-BFGS-B
  + \_prepare\_bounds(): Configures parameter search space

**optimization/sensitivity.py**

* **Purpose**: Parameter sensitivity analysis
* **Key Feature**:

python

Copy

def calculate\_sensitivity(self, params):

# Returns dict showing ∂(Error)/∂(Parameter)

**4. Visualization Files**

**visualization/plotter.py**

* **Key Plots**:

python

Copy

def plot\_stress\_paths(): # q-p' paths

def plot\_error\_history(): # Convergence

**visualization/live\_monitor.py**

* **Real-time Features**:
  + Updates plots every iteration
  + Shows current parameters/error

**5. Configuration Files**

**config.yaml**

* **Critical Sections**:

yaml

Copy

experimental\_data:

- type: "drained\_100"

filepath: "data/drained/100kPa.csv"

cell\_pressure: 100

parameters:

M\_tc:

base: [1.2, 1.8] # Bounds for intercept

slope: [-0.002, 0.002] # Bounds for pressure dependency

**6. Main Entry Point**

**main.py**

* **Execution Flow**:
  1. Loads config
  2. Initializes calibrator
  3. Runs optimization
  4. Saves results

**Key Adjustment Points for RS2**

**A. Modifying Result Extraction**

In rs2\_interface.py, enhance get\_stress\_strain():

python

Copy

def get\_stress\_strain(self):

results = self.model.GetResults()

return {

'strain': [s\*100 for s in results.GetColumn("Axial Strain")], # Convert to %

'stress': results.GetColumn("Deviator Stress"),

'pore\_pressure': results.GetColumn("Pore Pressure"),

'volume\_strain': results.GetColumn("Volumetric Strain"), # Added

'cell\_pressure': self.model.Stages[0].CellPressure

}

**B. Customizing Model Creation**

To adjust RS2 model properties:

python

Copy

def create\_test\_model(self, material, parameters, cell\_pressure, drainage):

mat = self.model.MaterialProperties.AddMaterial()

mat.Behavior = MaterialType.NorSand

# Set NorSand parameters

mat.NorSand.Mtc = parameters['M\_tc']

mat.NorSand.H0 = parameters['H\_0']

# Configure test

stage = self.model.Stages[0]

stage.CellPressure = cell\_pressure

stage.StrainRate = 0.1 # %/min

stage.NumberOfSteps = 50

**C. Adding New Material Models**

1. In material\_model.py:

python

Copy

class HardeningSoil(MaterialModel):

def \_\_init\_\_(self):

super().\_\_init\_\_("HardeningSoil")

self.parameters = {

'E50': [10000, 50000], # Stiffness

'm': [0.5, 1.0] # Power law exponent

}

1. Update rs2\_interface.py:

python

Copy

if material == "HardeningSoil":

mat.Behavior = MaterialType.HardeningSoil

mat.HardeningSoil.E50 = parameters['E50']

**Suggested Development Process**

1. **Test with Mock Data**:

python

Copy

# In rs2\_interface.py

class RS2Interface:

def \_\_init\_\_(self, test\_mode=False):

self.test\_mode = test\_mode

if not test\_mode:

import clr

# ... real RS2 setup ...

1. **Gradual RS2 Integration**:
   * Start with simple models
   * Add complexity incrementally
   * Verify results at each step
2. **Debugging Tips**:

python

Copy

# Print RS2 model summary

print(f"Current model: {self.model.GetSummary()}")

# Save RS2 model file

self.model.Save("debug.fez")

**File Relationship Diagram**

Copy

main.py → calibrator.py → rs2\_interface.py (RS2 API)

→ pressure\_optimizer.py

→ plotter.py

Would you like me to provide specific examples for:

1. Implementing a new material model?
2. Customizing the triaxial test configuration?
3. Adding additional result metrics (e.g., shear bands)?