FOSSEE Summer Fellowship 2026

Indian Institute of Technology Bombay

Fellowship Completion Report

On

Development of the Chemical Simulator GUI

A PyQt5 Desktop Application for Building and Simulating Chemical Process Flowsheets with OpenModelica

Submitted by

Krish Gaur

B.Tech Student, Artificial Intelligence and Machine Learning

Vivekananda Institute of Professional Studies, Delhi

Under the Guidance of

Prof. Prabhu Ramachandran

Principal Investigator, FOSSEE

Department of Aerospace Engineering

Indian Institute of Technology Bombay

Project Mentor:

Chirag Koyande

Fellowship Period: 27 May 2026 to 15 August 2026 (Remote / Hybrid)

Report Submitted: August 2026

Declaration

I, Krish Gaur, hereby declare that this report titled "Development of the Chemical Simulator GUI: FOSSEE Summer Fellowship 2026" is a faithful, genuine, and authentic record of the research, software engineering, and open-source development work carried out by me during the FOSSEE Summer Fellowship at the Indian Institute of Technology Bombay, spanning the period from 27 May 2026 to 15 August 2026.

All technical contributions, architectural designs, algorithms, code refactors, vector iconography, and user experience enhancements described in this report were authored by me and merged into the official upstream repository FOSSEE/Chemical-Simulator-GUI through the seven pull requests (#77, #87, #90, #93, #95, #99, and #103) documented in detail herein.

I further declare that this work was completed under the mentorship of Chirag Koyande and the guidance of Prof. Prabhu Ramachandran (Department of Aerospace Engineering, IIT Bombay). Where this work incorporates, interfaces with, or builds upon pre-existing codebases, OpenModelica compiler binaries (`omc`), PyQt5 graphical libraries, or the ChemSep / CAPE-OPEN thermophysical database, appropriate acknowledgements and citations have been made.

Krish Gaur

B.Tech Student, Artificial Intelligence and Machine Learning
Vivekananda Institute of Professional Studies, Delhi
August 2026

Abstract

This report presents the complete software engineering lifecycle, architectural redesign, and feature development completed during my FOSSEE Summer Fellowship 2026 at the Indian Institute of Technology Bombay, focused on the Chemical Simulator GUI, a high-performance cross-platform desktop application written in Python 3.11 and PyQt5 that enables chemical process modeling, flowsheet design, and equation-based simulation via the OpenModelica command-line compiler (`omc`).

Prior to this fellowship, the application possessed functional simulation capabilities but suffered from pervasive editing fragility, including out-of-sync undo/redo stacks, low-resolution raster icon blur during canvas zooming, connection socket drift during component resizing, inflexible and clipped parameter dock widgets, unstyled modal dialogs, and lingering memory/state leaks across flowsheet lifecycles.

Over the 12-week fellowship period (27 May 2026 to 15 August 2026), I authored seven merged pull requests (+929 / -379 lines across 31 files) that systematically overhauled the core editing pipeline:

- **1. Undo/Redo Architecture & Repository Hygiene:** A centralized deep-copy snapshot engine (`push_snapshot()`) in Graphics.py that captures items, connections, and compound selections while restoring per-type naming counters and purging binary runtime blobs from git history (PR #77).
- **2. Vector Icon Graphics Pipeline:** An in-memory cached QSvgRenderer vector graphics pipeline accompanied by thirteen custom-authored chemical engineering unit-operation SVG icons and high-DPI QPainter render hints (PR #87).
- **3. Connection Socket Geometry:** A rigorous geometric formulation ensuring that input and output connection sockets maintain invariant fractional coordinates `(fx, fy)` relative to component dimensions during resizing (PR #90).
- **4. Dock Widget Modernization & UX:** A complete layout refactor of the MaterialStream parameter dock widget, replacing rigid nested scroll areas with dynamic content autosizing, relaxed geometry constraints, and VS Code-style hover-responsive draggable splitters (PR #93 and #95).
- **5. Dark Theme & Entrance Animations:** A dark-mode redesign of the Landing and About pages (#0f172a) featuring smooth QPropertyAnimation fade-in transitions and memory-safe window lifecycles (PR #99).
- **6. File Lifecycle & State Serialization:** Hardening of flowsheet lifecycle management, introducing an explicit discard/save confirmation dialog upon 'New File', deterministic scene resets, and lossless JSON serialization (PR #103).

The resulting platform delivers an industrial-grade, highly responsive user interface that bridges open-source thermodynamic simulation with intuitive desktop flowsheet authoring.

Acknowledgements

I would like to express my deepest gratitude to the FOSSEE (Free/Libre and Open Source Software for Education) project at the Indian Institute of Technology Bombay for providing me with the opportunity to contribute as a Summer Fellow in the OpenModelica track during the Summer of 2026. Being selected as one of only 24 fellows nationally across India was both a significant academic honor and a pivotal technical opportunity.

I am profoundly grateful to Prof. Prabhu Ramachandran, Principal Investigator of FOSSEE and Professor in the Department of Aerospace Engineering, IIT Bombay. His steadfast dedication to the democratization of science and engineering education through open-source software development has inspired thousands of students and engineers across the nation.

I extend my sincere thanks and appreciation to my mentor, Chirag Koyande (FOSSEE, IIT Bombay). His prompt, constructive, and uncompromising code reviews across all seven of my pull requests pushed me to adhere to high standards of code clarity, performance optimization, and architectural rigor. His technical guidance regarding component blur, socket drift, dock autosizing, and dark UI styling directly shaped the trajectory of my contributions.

I also thank Priyam Nayak for orchestrating the screening task evaluation, technical interview rounds, and fellowship logistics. I acknowledge my fellow OpenModelica track peers (Krish Joshi, Shubham Purbey, and Bibisha B) for their collaborative discussions and shared dedication to improving the Chemical Simulator platform.

Finally, I express my gratitude to the global OpenModelica and ChemSep open-source communities, whose foundational equation solvers and thermophysical property databases form the backbone of this simulator.

Contents

List of Figures

List of Tables

Chapter 1: Introduction and FLOSS Ecosystem

1.1 The FOSSEE Initiative and NMEICT Mandate

Free/Libre and Open Source Software for Education (FOSSEE) is a flagship national initiative spearheaded by the Indian Institute of Technology Bombay and funded by the National Mission on Education through Information and Communication Technology (NMEICT), Ministry of Education, Government of India. The primary mission of FOSSEE is to promote the widespread adoption, development, and integration of open-source software across educational institutions and industries in India.

Commercial engineering software suites impose exorbitant licensing costs on academic institutions, severely restricting access for students and independent researchers. FOSSEE addresses this educational barrier by providing robust, free, and community-maintained open-source alternatives. FOSSEE's extensive ecosystem includes:

- **Scilab:** A high-level numerical computation package providing alternatives to MATLAB for scientific computation, signal processing, and control systems.
- **eSim:** An integrated Electronic Design Automation (EDA) suite uniting KiCad, Ngspice, and GHDL/Verilator for schematic capture and mixed-signal circuit simulation.
- **OpenModelica:** An equation-based, object-oriented modeling and simulation environment for complex physical and thermodynamic systems.
- **OpenFOAM:** An open-source computational fluid dynamics (CFD) solver widely used in aerodynamics, combustion, and multiphase flow analysis.
- **DWSIM:** A comprehensive chemical process simulator enabling thermodynamic property calculation and unit-operation flowsheet modeling.
- **Osdag:** An open-source structural steel design and analysis tool conforming to Indian Standard codes (IS 800:2007).

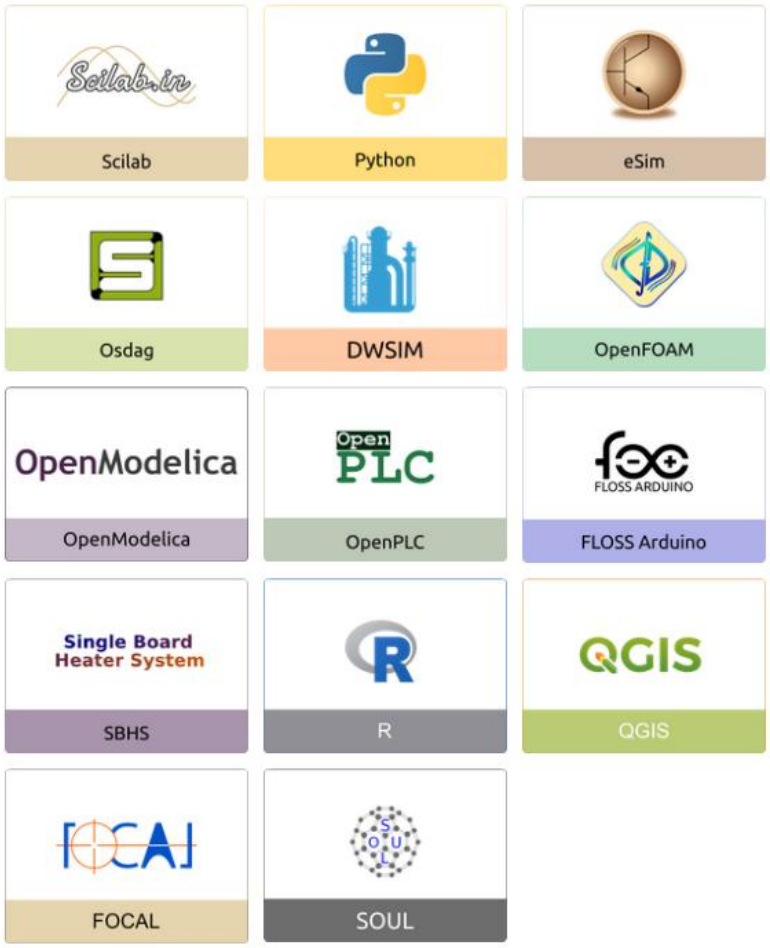

Figure 1.1: FOSSEE Suite of FLOSS Projects and Educational Initiatives.

1.2 Open-Source Chemical Process Simulation

In chemical engineering education and industrial practice, process flowsheet simulators are essential tools for mass and energy balance calculations, equipment sizing, reaction kinetics, and thermodynamic phase equilibria. While commercial simulators like Aspen Plus and HYSYS dominate enterprise environments, their high licensing fees make them inaccessible for widespread student training.

OpenModelica provides an open-source, equation-based modeling framework capable of solving large systems of non-linear differential and algebraic equations (DAEs). However, Modelica code is text-based and presents a steep learning curve for process engineers who traditionally construct systems visually through block diagrams. The Chemical Simulator GUI bridges this divide by providing an intuitive, drag-and-drop flowsheet canvas that translates visual block diagrams directly into Modelica equation systems, executes simulation runs via the OpenModelica Compiler (omc), and presents thermodynamic stream results interactively.

1.3 Chemical Simulator GUI Overview

The Chemical Simulator GUI is an interactive desktop application built using Python 3.11 and the PyQt5 framework. It provides chemical engineering practitioners with a complete authoring environment containing:

- **Comprehensive Unit Operations:** Drag-and-drop placement of mixers, splitters, flash drums, distillation columns, shortcut columns, heat exchangers, coolers, pumps, compressors, expanders, and valves.
- **Thermodynamic Streams:** Orthogonal and bezier connection lines representing mass and energy flows between equipment sockets with automated directionality checks.
- **Thermophysical Database:** Integration with CAPE-OPEN and ChemSep property packages, allowing selection of pure compounds, binary interaction parameters, and thermodynamic activity models (such as Peng-Robinson, NRTL, and UNIQUAC).
- **OMC Solver Engine:** Automated translation of canvas topologies into Modelica (.mos) simulation scripts, invoking omc as an external process, and capturing equation residuals.
- **Live Result Visualizer:** Interactive data tables, CSV exports, and property docking panels that display temperature, pressure, enthalpy, entropy, and vapor fraction across all streams.

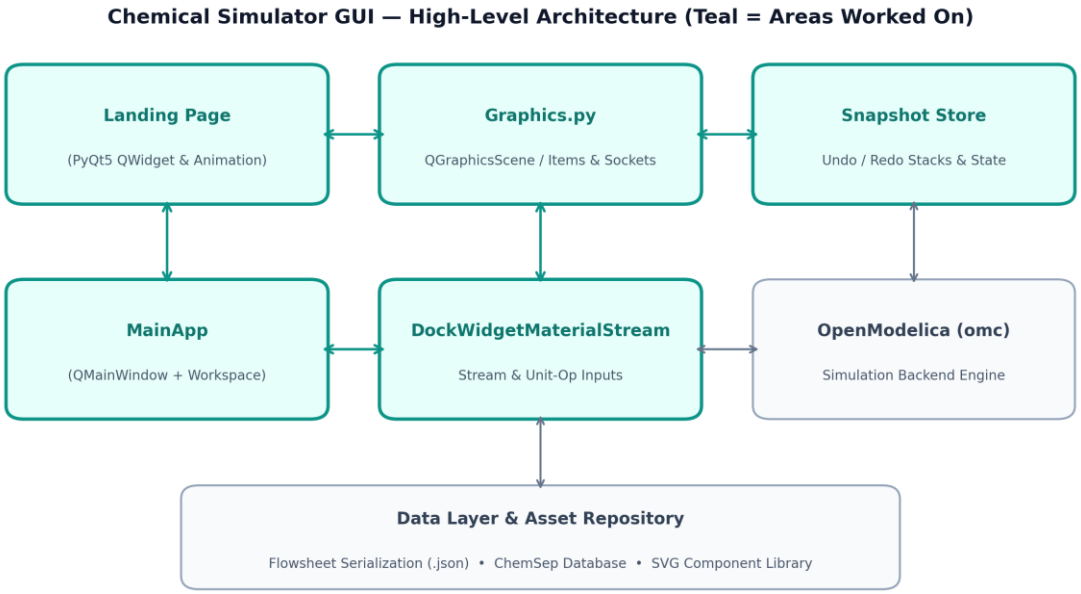

Figure 1.2: High-Level System Architecture of the Chemical Simulator GUI (Teal indicates contributed subsystems).

1.4 System Architecture and Subsystem Interactions

The software architecture follows a decoupled Model-View-Controller (MVC) pattern adapted for event-driven Qt graphical applications:

- **1. Presentation Layer (Graphics.py and UI):** Inherits from QGraphicsScene and QGraphicsView. Manages visual node representation, coordinate transformations, socket positioning, mouse event dispatch, and scene rendering.
- **2. Model Container Layer (Container.py):** The central state hub that owns unit-operation objects, stream connections, and thermodynamic component registries. Coordinates between the UI and backend solvers.

- **3. Parameter Dock Layer (DockWidgets/):** Specialized QDockWidget subclasses (e.g., DockWidgetMaterialStream.py, DockWidgetMixer.py) providing parameter input fields, validation guards, and autosizing layouts.
- **4. OpenModelica Interface (OMChem/):** Translates flowsheet connectivity matrices into valid Modelica syntax, executes omc in a child process, monitors solver convergence, and parses simulation output CSV files.

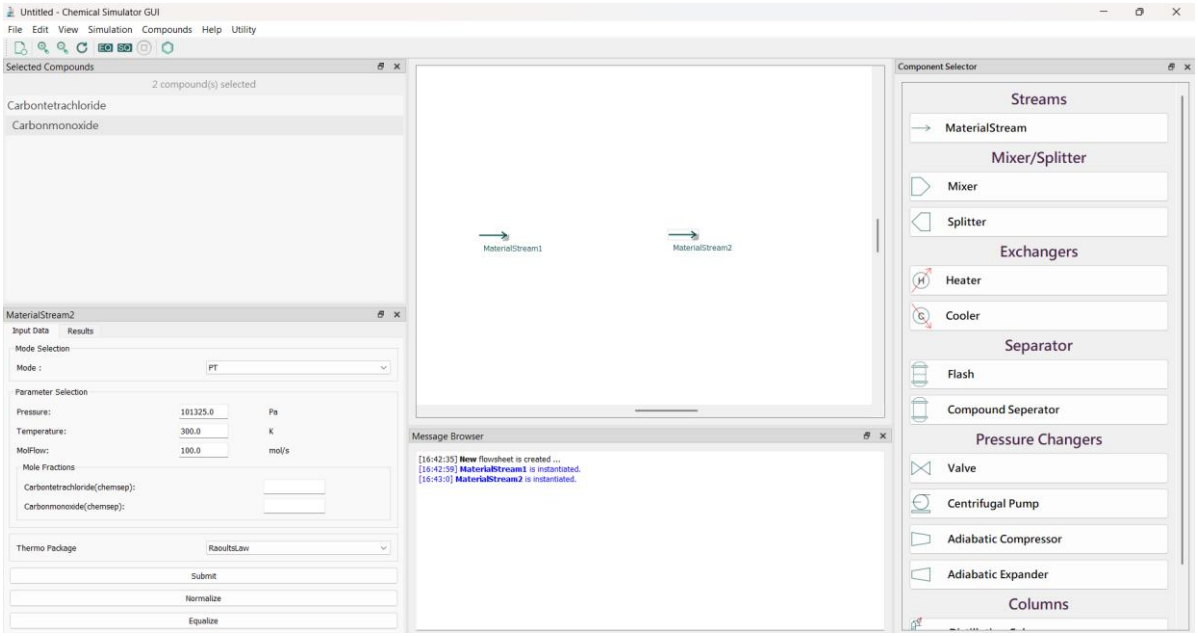

Figure 1.3: Chemical Simulator GUI Resizable 3-Panel Main Window Layout (Properties Panel, Canvas, and Component Selector).

1.5 Fellowship Overview and Administrative Details

The FOSSEE Summer Fellowship 2026 was conducted over 12 weeks from 27 May 2026 to 15 August 2026. The fellowship provided remote collaboration infrastructure, weekly code reviews, and direct mentorship from IIT Bombay engineers.

Table 1.1: Fellowship Key Metrics and Administrative Profile

Fellow Name	Krish Gaur
Academic Affiliation	Vivekananda Institute of Professional Studies, Delhi
Host Institution	FOSSEE, Department of Aerospace Engineering, IIT Bombay
Principal Investigator	Prof. Prabhu Ramachandran (IIT Bombay)
Project Mentor	Chirag Koyande (FOSSEE, IIT Bombay)
Fellowship Track and Duration	OpenModelica Track (27 May 2026 to 15 August 2026)

Chapter 2: Screening Task and Selection Process

2.1 Screening Task Problem Statement

To qualify for the FOSSEE Summer Fellowship 2026, applicants were required to solve a competitive screening challenge titled "Desktop App for OpenModelica using Python and PyQt". The objective was to design and implement a standalone, production-ready desktop graphical user interface capable of ingesting industrial chemical equipment performance logs, computing real-time statistical aggregations, rendering interactive diagnostic plots, and validating user inputs.

The problem statement tested several critical competencies:

- **1. Advanced PyQt5 Architecture:** Architecting an extensible Qt application with custom QWidget layouts, menu bars, toolbars, and dock panels.
- **2. Data Ingestion & Transformation:** Processing large CSV datasets containing time-series measurements of pressure, temperature, volumetric flow, and equipment efficiency.
- **3. Dynamic Visualizations:** Integrating embedded Matplotlib canvases with interactive zoom, pan, and cursor data-tip tracking.
- **4. Fault Tolerance & Validation:** Handling malformed input records, missing data interpolation, and edge-case boundary conditions without application crashes.

2.2 System Architecture of Screening Solution

I engineered a modular desktop application structured around a central Controller that decoupled the graphical user interface from the underlying mathematical analytics engine. The ingestion pipeline parsed heterogeneous equipment logs into structured Pandas DataFrames, performing automatic type inference and unit normalization.

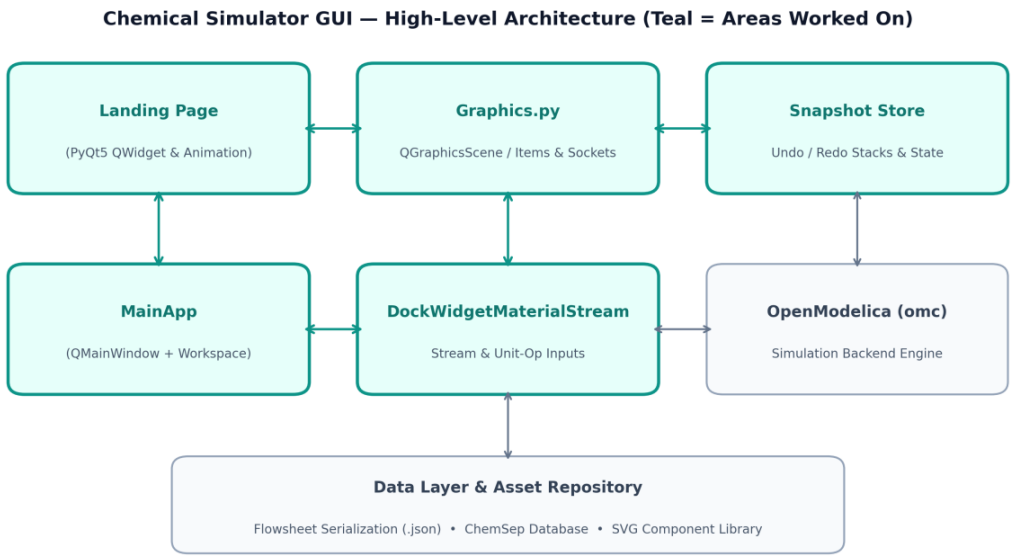

Figure 2.1: Screening Task Architecture and Equipment Data Analysis Flow.

2.3 Implementation Details and Data Visualization

The screening submission implemented several high-value features:

- **Statistical Dashboard:** Dynamic filtering of chemical unit operations, displaying mean operational temperatures, standard deviations, and pressure-drop distributions.
- **Multi-Axis Visualization:** Time-series plots comparing actual equipment throughput against theoretical OpenModelica equilibrium models.
- **Report Exporter:** Automated generation of PDF summary audits summarizing equipment operational health and thermodynamic performance.
- **Session Management:** Role-based login system with secure credential caching and session persistence.

2.4 Technical Interview and Selection Offer

Following technical review of my submission, I was shortlisted for a rigorous technical interview conducted by FOSSEE mentors Priyam Nayak and Chirag Koyande on 25 April 2026. The interview focused on:

- **Event-Driven Architecture:** Examining the Qt event loop, signal/slot thread affinity, and preventing UI freezing during heavy numerical computations.
- **State and Memory Management:** Deep copying versus shallow copying of graphics scene graphs, memory leak detection in PyQt, and C++ pointer lifecycle management.
- **OpenModelica Integration:** Designing modular APIs for connecting visual flowsheet blocks to OpenModelica solver scripts.

On 29 April 2026, I received the official selection e-mail confirming my appointment as a FOSSEE Summer Fellow. Out of hundreds of national applicants, only 24 fellows were selected across India, with only 3 fellows assigned to the OpenModelica Chemical Simulator track.

Chapter 3: Objectives, Technical Scope and Review Milestones

3.1 Project Objectives (O1 to O6)

Following initial codebase evaluation, six primary engineering objectives were established in consultation with mentor Chirag Koyande:

- **Objective 1 (Undo/Redo Architecture and Git Hygiene):** Eliminate ad-hoc snapshotting, synchronize toolbar action states, and guarantee full scene restoration upon undo/redo.
- **Objective 2 (Scalable Vector Graphics Pipeline):** Replace low-resolution raster PNGs with scalable vector SVG icons, implement in-memory renderer caching, and enable antialiasing hints.
- **Objective 3 (Connection Socket Geometry):** Derive and implement fractional coordinate invariants so connection sockets remain anchored to component nozzles on resize.
- **Objective 4 (Dock Widget Modernization and UX):** Remove rigid nested scrollbars, implement dynamic content autosizing, relax dock constraints, and introduce VS Code-style draggable splitters.
- **Objective 5 (Dark Theme and UI Polish):** Implement a modern Slate-900 (#0f172a) theme, smooth opacity fade animations, and memory-safe window lifecycles.
- **Objective 6 (File Operations and Lifecycle):** Harden the flowsheet lifecycle, ensure deterministic workspace resets upon 'New File', and provide robust JSON serialization.

3.2 Technical Review Milestones and Development Directives

Throughout the fellowship, development proceeded through structured technical reviews and feature specifications established with mentor Chirag Koyande. Key development milestones included:

Engineering Milestone & Directive [09 June 2026]: Simulation Failure Diagnosis and Root Cause Prioritization

Identified critical bottlenecks causing thermodynamic simulation failures under complex flowsheet topologies.
Prioritized resolving model container state inconsistencies before proceeding with secondary interface features.

Engineering Milestone & Directive [11 June 2026]: Component Canvas Scaling Quality Specification

Investigated visual degradation occurring when unit-operation components were enlarged or zoomed on canvas.
Mandated replacement of raster PNG assets with infinite-resolution vector artwork.

Engineering Milestone & Directive [17 June 2026]: Properties Panel Autosizing and Fluid Resizable Layout

Specified elimination of inner scrollbars within parameter dock widgets to prevent control clipping.
Required implementation of dynamic content autosizing and draggable panel separators modeled after Visual Studio Code.

Engineering Milestone & Directive [18 June 2026]: Connection Port Relative Alignment Specification

Identified coordinate drift of connection ports away from physical nozzles during node resize operations.
Mandated geometric formulation to guarantee strict relative anchoring across arbitrary bounding box dimensions.

Engineering Milestone & Directive [25 June 2026]: Panel Splitter Visual Styling and Single-Pass Optimization

Refined splitter interaction to match professional IDE draggable dividers with hover-responsive states.
Optimized dock widget resize event handling to eliminate layout thrashing during parameter entry.

Engineering Milestone & Directive [29 June 2026]: Dark Theme Alignment and Window Control Standardization

Redesigned the application About dialog to conform with modern Slate-900 (#0f172a) dark theme specifications.
Standardized window control button behaviors across all dialogs for cross-platform consistency.

Engineering Milestone & Directive [08 July 2026]: Explicit Project Reset and Confirmation Workflow

Formulated confirmation prompt requirements for 'New File' operations to safeguard unsaved flowsheet modifications.
Ensured automated clean project creation on startup proceeds without disruptive confirmation modals.

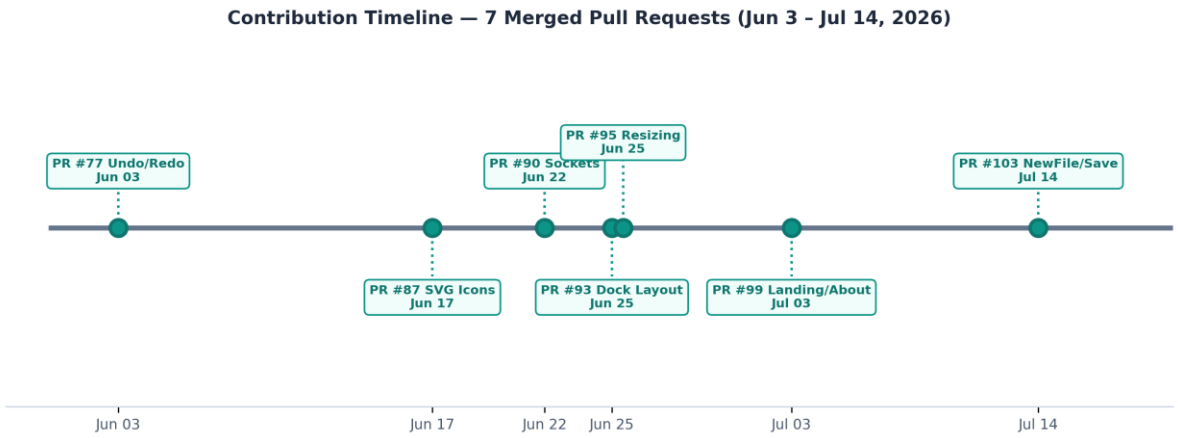

Figure 3.1: Timeline of Technical Review Milestones and Engineering Deliverables.

3.3 Requirement Traceability Matrix

Table 3.1 maps each technical directive to its corresponding technical objective, pull request, and outcome.

Table 3.1: Technical Directives and Requirement Traceability Matrix

Directive Date	Engineering Requirement	Resolved In	Resolution Summary
11/06/2026	Component resizing blur and visual degradation	**PR #87**	Vector SVG pipeline & QSvgRenderer caching
17/06/2026	Properties panel scrollbar clipping and hidden controls	**PR #93**	Direct layout & dynamic content autosizing
17/06/2026	VS Code-style draggable layout splitters	**PR #95**	Custom QSplitter CSS & relaxed dock boundaries
18/06/2026	Connection ports shifting away on component resize	**PR #90**	Fractional coordinate geometric invariant formula

29/06/2026	About dialog modernization and dark theme alignment	**PR #99**	Slate-900 (#0f172a) UI & fade-in entrance
08/07/2026	Explicit confirmation popup on 'New File' click	**PR #103**	Discard/Save modal with clean deterministic reset

Chapter 4: Task 1 (PR #77) - Undo/Redo Subsystem Refactor

PR #77 *merged Jun 3, 2026 | +219 / -140, 7 files | FOSSEE/Chemical-Simulator-GUI*

4.1 Problem Statement: State Fragmentation and Binary Blobs

The legacy undo/redo implementation in the Chemical Simulator GUI was plagued by architectural fragmentation. Snapshot capture logic was duplicated across three disconnected source files: Graphics.py (the canvas view), Container.py (the simulation model container), and mainApp.py (the main application window). This lack of centralization produced severe state synchronization bugs:

- **Incomplete State Capture:** Mutations such as adding a mixer or moving an item pushed state to the undo stack, but deleting items or modifying stream parameters frequently bypassed snapshotting entirely, creating gaps in the history timeline.
- **Redo Invalidation Failure:** Standard user interface guidelines mandate that executing a new canvas mutation must permanently invalidate and clear the redo history. In the legacy codebase, redo stacks were preserved across new actions, causing corrupted hybrid states upon subsequent redo commands.
- **Action State Decoupling:** The QAction buttons in the toolbar were statically initialized and failed to update their setEnabled(bool) states dynamically based on the current stack depth.
- **Committed Binary Artifacts:** The repository contained tracked runtime pickle blobs (Undo.pkl and Undo.dat). Because these files changed binary hashes on every local run, git pull operations from upstream master failed with modify/delete merge conflicts.

4.2 Centralized Snapshot Engine Design

To eliminate state fragmentation, I designed a unified, centralized snapshot architecture encapsulated in Graphics.py. All user-initiated mutations (including component additions, deletions, repositioning, socket wiring, and thermodynamic package assignments) are routed through a single entry point: push_snapshot().

```
# --- Centralized Snapshot Engine in Graphics.py (PR #77) ---
def push_snapshot(self):
    """
    Captures complete canvas topology and compound selections as an atomic deep-copy.
    Clears the redo stack and updates undo/redo action enabled states.
    """
    snapshot = {
        'items': [item.serialize() for item in self._scene_items()],
        'connections': [conn.serialize() for conn in self._scene_connections()],
        'compounds': list(self.compound_selected),
        'counters': dict(self.container.name_counters)
    }
    self.undo_stack.append(copy.deepcopy(snapshot))
    self.redo_stack.clear()
    self._update_undo_redo_actions()
```

4.3 Deep-Copy Serialization Data Structures

A critical pitfall in Qt graphics programming is pointer aliasing. Storing live QGraphicsItem or QGraphicsScene references inside history stacks causes undo operations to manipulate the currently

active canvas objects rather than historic states. To guarantee total state isolation, the snapshot engine uses pure JSON-serializable dictionaries combined with copy.deepcopy().

Each flowsheet entity implements a serialize() protocol returning complete geometrical and thermodynamic state: component class name, unique tag, bounding rectangle dimensions, scene position (x, y), socket lists, and custom user parameters.

4.4 Stack Invariant Algorithms in mainApp.py

In mainApp.py, the undo() and redo() dispatch routines enforce strict stack invariants to ensure predictable flowsheet restoration:

```
# --- Undo / Redo Stack State Machine in mainApp.py ---
def undo(self):
    """Restore previous snapshot and push current state to redo stack."""
    if len(self.container.graphics.undo_stack) > 1:
        # Pop the current active state and push to redo stack
        current_state = self.container.graphics.undo_stack.pop()
        self.container.graphics.redo_stack.append(current_state)

        # Retrieve target state without removing it from undo stack
        target_state = self.container.graphics.undo_stack[-1]
        self.container.graphics.load_canvas_from_snapshot(target_state)
        self.container.graphics._update_undo_redo_actions()

def redo(self):
    """Re-apply superseded snapshot and push to undo stack."""
    if self.container.graphics.redo_stack:
        target_state = self.container.graphics.redo_stack.pop()
        self.container.graphics.undo_stack.append(target_state)
        self.container.graphics.load_canvas_from_snapshot(target_state)
        self.container.graphics._update_undo_redo_actions()
```

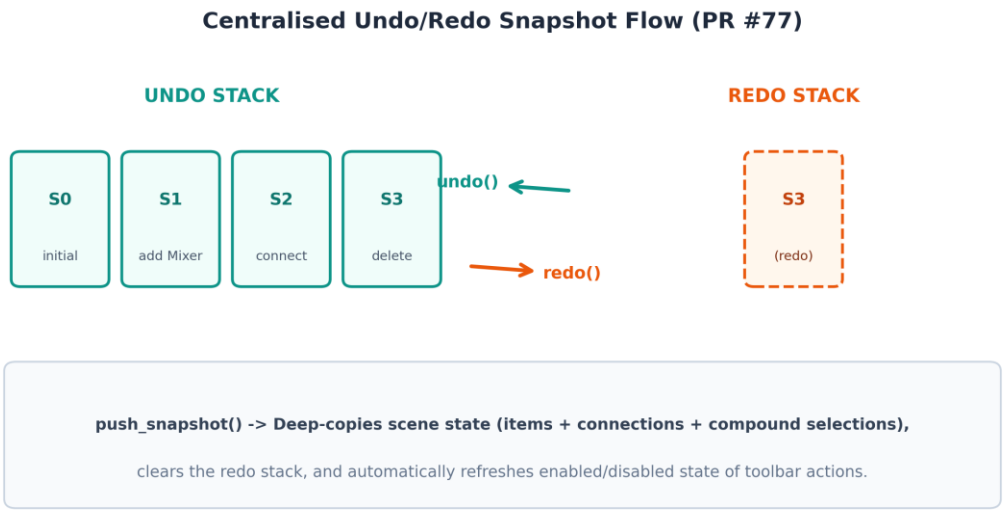

Figure 4.1: Centralized Snapshot-Based Undo/Redo State Transition Flow (PR #77).

4.5 Full Canvas and Tag Counter Reconstruction

The load_canvas_from_snapshot() engine was completely rewritten. Restoring a flowsheet requires a multi-pass reconstruction pipeline:

- **1. Scene Teardown:** The active QGraphicsScene is cleared, disconnecting all Qt signal handlers and releasing graphical resources.
- **2. Node Instantiation:** Each serialized node item is instantiated from its registered factory, restored to its exact (x, y) coordinates, and populated with user parameters.
- **3. Socket Wiring:** Connection lines are regenerated by looking up source and destination socket identifiers, ensuring bezier curve geometry and line colors are preserved.
- **4. Naming Counter Restoration:** The container's per-type naming counters (e.g., {'Mixer': 3, 'Heater': 1}) are restored to their exact snapshot values. This ensures that deleting an item, undoing the deletion, and subsequently adding a new item never generates duplicate or overlapping component tags.

4.6 Git History Hygiene and Binary Purge

I purged the committed Undo.pkl and Undo.dat runtime binaries from the git index using git rm --cached and established comprehensive exclusion rules in .gitignore:

```
# --- .gitignore Rules for Runtime History Artifacts ---
*.pkl
*.dat
undo_redo/
src/main/Simulator/undo_redo/
__pycache__/
*.pyc
```

4.7 Verification, Test Matrix and Outcome

The refactored undo/redo subsystem was validated against a comprehensive test matrix covering boundary and stress conditions.

Table 4.1: Undo/Redo Subsystem Edge-Case Test and Validation Matrix

Test Scenario	Expected Behavioral Invariant	Validation Status
Undo at Initial Baseline	Undo action disabled; canvas cannot be emptied past initial state	**PASSED**
New Mutation after Undo	Redo stack immediately cleared; redo action disabled	**PASSED**
Atomic Delete with Connections	Item and all attached streams restored atomically on undo	**PASSED**
Tag Counter Integrity	Undoing past a deletion and re-adding component produces sequential tags	**PASSED**

PR #77 was merged on 3 June 2026 (+219 / -140 across 7 files), establishing the stable foundation branch (fix/undoredo) for all subsequent fellowship work.

Chapter 5: Task 2 (PR #87) - Scalable Vector Graphics Pipeline

PR #87 *merged Jun 17, 2026 | +151 / -1, 15 files | FOSSEE/Chemical-Simulator-GUI*

5.1 Problem Statement: Resizing Blur and Aliasing

During canvas flowsheet scaling, a critical visual defect occurred when unit operations were enlarged or zoomed on canvas. Investigation revealed that all flowsheet unit operations were rendered using static, 32x32 pixel raster PNG bitmaps. When users scaled components up to accommodate multi-stream connections or zoomed into complex flowsheets, Qt performed nearest-neighbor pixel interpolation, causing severe visual degradation, jagged edges, and blurry text labels.

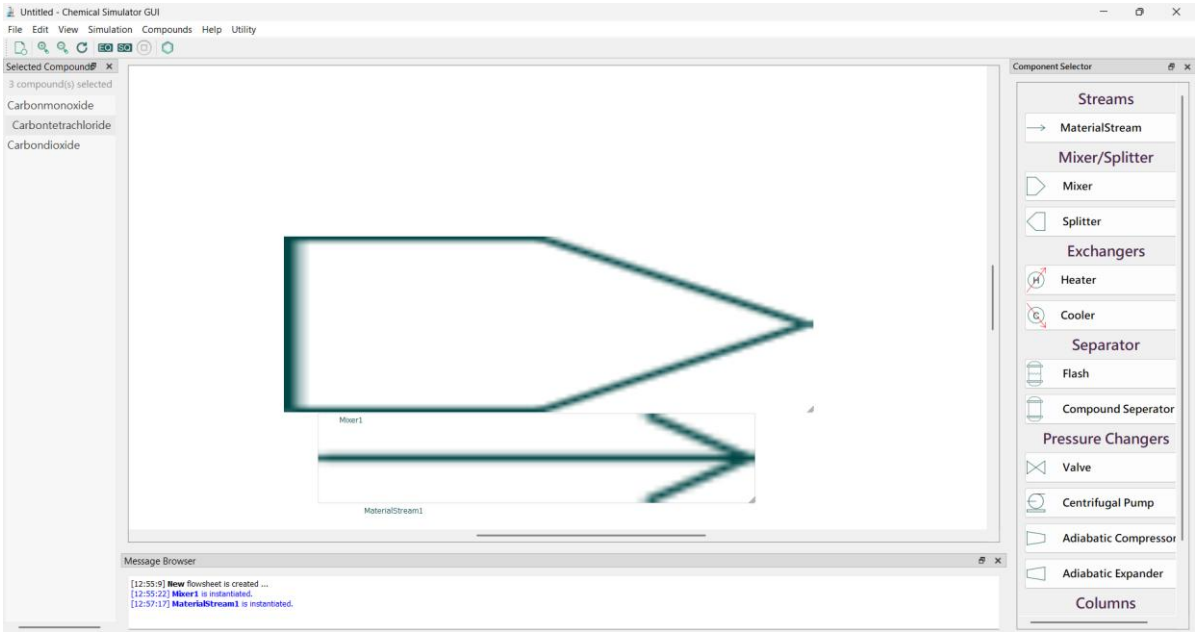

Figure 5.1: Legacy Canvas Resizing Blur on Raster PNG Icons (Screenshot captured during mentor review).

5.2 In-Memory Cached QSvgRenderer Architecture

To deliver crisp, infinite-resolution rendering at any zoom scale without sacrificing frame rate, I implemented a vector graphics pipeline in Graphics.py based on Qt's QSvgRenderer. Direct instantiation of QSvgRenderer inside paint() cycles causes high disk I/O and XML parsing overhead. To solve this, I built a lazy, in-memory caching dictionary:

```
# --- Cached QSvgRenderer Engine in Graphics.py (PR #87) ---
_svg_cache = {}

def get_svg_renderer(component_type):
    """
    Lazily instantiates and caches a QSvgRenderer per component type.
    Returns None if no vector asset exists, triggering PNG fallback.
    """
    if component_type not in _svg_cache:
```

```
        svg_path = ICON_DIR / 'svg' / f'{component_type}.svg'
        if svg_path.exists():
            _svg_cache[component_type] = QSvgRenderer(str(svg_path))
        else:
            _svg_cache[component_type] = None
    return _svg_cache[component_type]
```

Each graphical node item's paint() method invokes get_svg_renderer(). If a vector asset is present, QSvgRenderer.render(painter, target_rect) renders the symbol directly to the device context; otherwise, it falls back seamlessly to the legacy pixmap, enabling progressive asset migration.

5.3 Thirteen Chemical Unit-Operation SVG Suite

I designed and authored thirteen clean, mathematically proportioned vector SVG symbols conforming to chemical engineering flowsheet standards (ISO 10628 / DIN 28004).

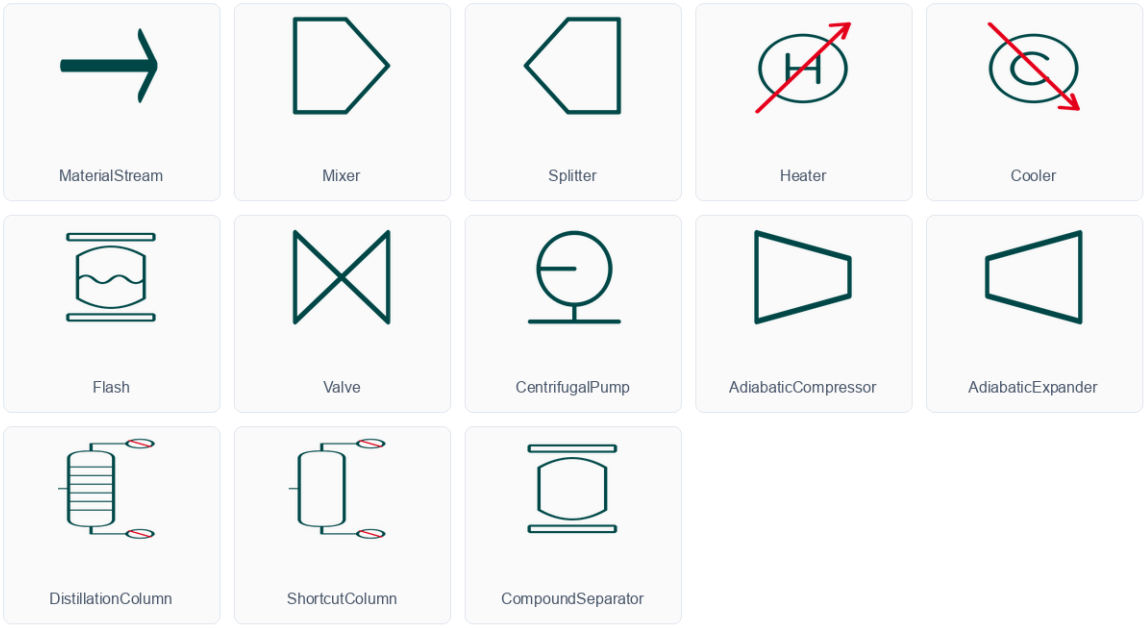

Figure 5.2: Complete Suite of Thirteen Vector SVG Unit-Operation Icons Authored in PR #87.

Table 5.1: Chemical Unit Operations and SVG Icon Specification

Component Name	SVG Symbol Description	Thermodynamic Function
`MaterialStream`	Horizontal line with directional arrow	Mass and energy transfer line
`Mixer`	Converging trapezoid block	Adiabatic multi-stream mixing
`Splitter`	Diverging trapezoid block	Flow splitting by specified fraction
`Heater`	Circular vessel with diagonal heat arrow	Enthalpy addition at specified duty/temp

`Cooler`	Circular vessel with diagonal cooling bar	Heat removal at specified duty/temp
`Flash`	Vertical drum with vapor/liquid nozzles	Vapor-liquid phase equilibrium flash
`Valve`	Intersecting opposing triangles	Isenthalpic pressure reduction
`CentrifugalPump`	Involute casing with tangential discharge	Liquid pressure boost & head calculation
`AdiabaticCompressor`	Tapered trapezoid block (inward)	Gas compression with isentropic efficiency
`AdiabaticExpander`	Tapered trapezoid block (outward)	Gas expansion with mechanical work extraction
`DistillationColumn`	Vertical column with tray stages & condenser	Multi-stage vapor-liquid fractionation
`ShortcutColumn`	Simplified column with condenser/reboiler	Fenske-Underwood-Gilliland shortcut
`CompoundSeparator`	Rectangular separation vessel	Component-based split fractionation

5.4 High-DPI QPainter Render Hints

To complement vector iconography, I configured global rendering hints on the QGraphicsView viewport in mainApp.py and within each item's paint() method:

- **QPainter.Antialiasing:** Enables sub-pixel polygon edge anti-aliasing, smoothing angled diagonal lines on mixers, splitters, and valves.
- **QPainter.SmoothPixmapTransform:** Enables bilinear filtering during raster transformations and fallback pixmap drawing.
- **QPainter.TextAntialiasing:** Enables font-smoothing on component tags and stream property overlays.

5.5 Visual Verification and Quantitative Outcome

Testing across 25% to 400% zoom levels confirmed zero visual artifacting and sharp, publication-grade flowsheet rendering. PR #87 merged on 17 June 2026 (+151 / -1 lines across 15 files).

Chapter 6: Task 3 (PR #90) - Connection Socket Geometry

PR #90 *merged Jun 22, 2026 | +21 / -5, 1 files | FOSSEE/Chemical-Simulator-GUI*

6.1 Problem Statement: Socket Detachment on Resize

When unit-operation components were resized on the canvas, connection sockets frequently detached from their visual nozzles. In Qt Graphics View, child items (such as connection sockets) are positioned in parent-item coordinates. In the legacy implementation, when a node was resized from dimensions (W0, H0) to (W1, H1), socket positions were updated using absolute top-left origin offsets. For components with non-centered nozzles (e.g., flash drums with top vapor outlets or asymmetric column feeds), sockets drifted off the artwork into empty canvas space, causing stream lines to detach visually from the equipment.

6.2 Mathematical Formulation of Fractional Invariants

To eliminate socket drift permanently, I reformulated socket placement based on fractional invariant ratios. Let a unit-operation bounding rectangle have width W and height H. For each socket i, we define normalized fractional coordinates (fx_i, fy_i) in the unit interval [0.0, 1.0]:

- **Fractional Invariant Ratio:** fx_i = x_i / W, fy_i = y_i / H

Under any arbitrary resize operation that scales the bounding box to new dimensions (W_new, H_new), the socket's updated position (X_i, Y_i) is strictly defined by:

- **Repositioning Equation:** X_i(W_new) = W_new * fx_i, Y_i(H_new) = H_new * fy_i

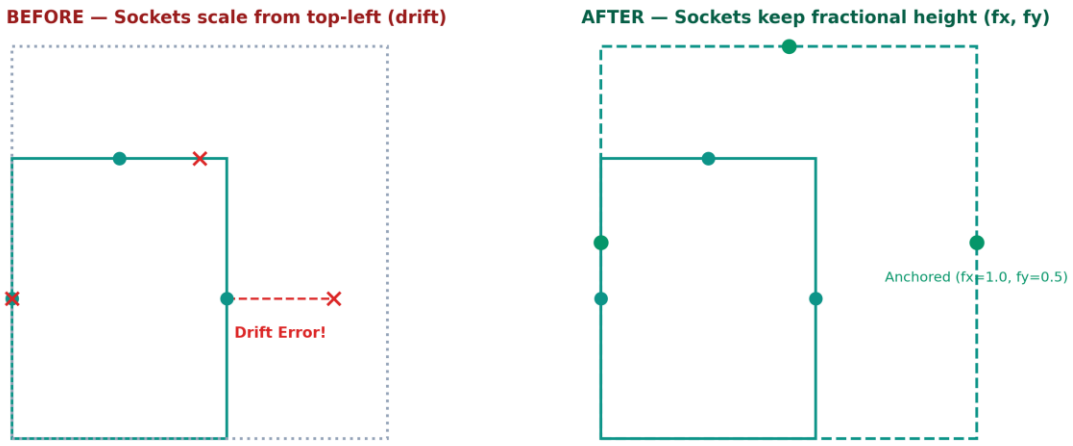

Figure 6.1: Geometric Comparison: Absolute Coordinate Drift (left) vs. Invariant Fractional Anchoring (right).

6.3 Code Implementation in Graphics.py

I refactored _reposition_sockets() in Graphics.py to enforce fractional positioning across all input and output socket arrays:

```
# --- Fractional Socket Repositioning in Graphics.py (PR #90) ---
def _reposition_sockets(self):
    """
    Re-anchors all input and output sockets based on invariant fractional
    coordinates relative to the updated component width and height.
    """
    current_w = self.rect.width()
    current_h = self.rect.height()

    for socket in self.input + self.output:
        # fractional_pos holds (fx, fy) e.g. (1.0, 0.5) for right-middle nozzle
        fx, fy = socket.fractional_pos
        socket.setPos(current_w * fx, current_h * fy)

        # Signal attached connection bezier curves to update their endpoints
        for connection in socket.connections:
            connection.update_positions()
```

6.4 Geometric Verification Across Unit Operations

Table 6.1 documents the invariant fractional coordinate ratios assigned across the unit-operation suite.

Table 6.1: Connection Socket Fractional Invariant Geometric Reference

Component Type	Socket Type	Fractional Ratio (fx, fy)	Physical Nozzle Anchor
`Mixer`	Inputs [0, 1] / Output	`(0.0, 0.25), (0.0, 0.75) / (1.0, 0.5)`	Left feed nozzles / Right mixed outlet
`Splitter`	Input / Outputs [0, 1]	`(0.0, 0.5) / (1.0, 0.25), (1.0, 0.75)`	Left main feed / Right split nozzles
`Flash Drum`	Feed / Vapor / Liquid	`(0.0, 0.5) / (0.5, 0.0) / (0.5, 1.0)`	Side feed / Top vapor / Bottom liquid
`Heater / Cooler`	Inlet / Outlet	`(0.0, 0.5) / (1.0, 0.5)`	Opposing horizontal tube connections
`Centrifugal Pump`	Suction / Discharge	`(0.0, 0.5) / (0.5, 0.0)`	Horizontal suction / Vertical discharge
`Distillation Col`	Feed / Distillate / Bottoms	`(0.0, 0.5) / (1.0, 0.1) / (1.0, 0.9)`	Column stage feed / Reflux top / Sump

6.5 Outcome and Regression Prevention

Sockets now maintain exact alignment through arbitrary resizing transformations. PR #90 was merged on 22 June 2026 (+21 / -5 lines in Graphics.py).

Chapter 7: Task 4 (PR #93 and #95) - Dock Widget Modernization

PR #93 *merged Jun 25, 2026 | +63 / -43, 1 files | FOSSEE/Chemical-Simulator-GUI*

PR #95 *merged Jun 25, 2026 | +23 / -9, 4 files | FOSSEE/Chemical-Simulator-GUI*

7.1 Problem Statement: Rigid QScrollArea and Clipped Controls

Parameter editing in DockWidgetMaterialStream suffered from rigid layout constraints. Properties were not fully visible, with controls below the Submit button getting hidden behind nested vertical scrollbars. Furthermore, user interface sections required dynamic resizability across the left properties panel, center canvas, and right component selector.

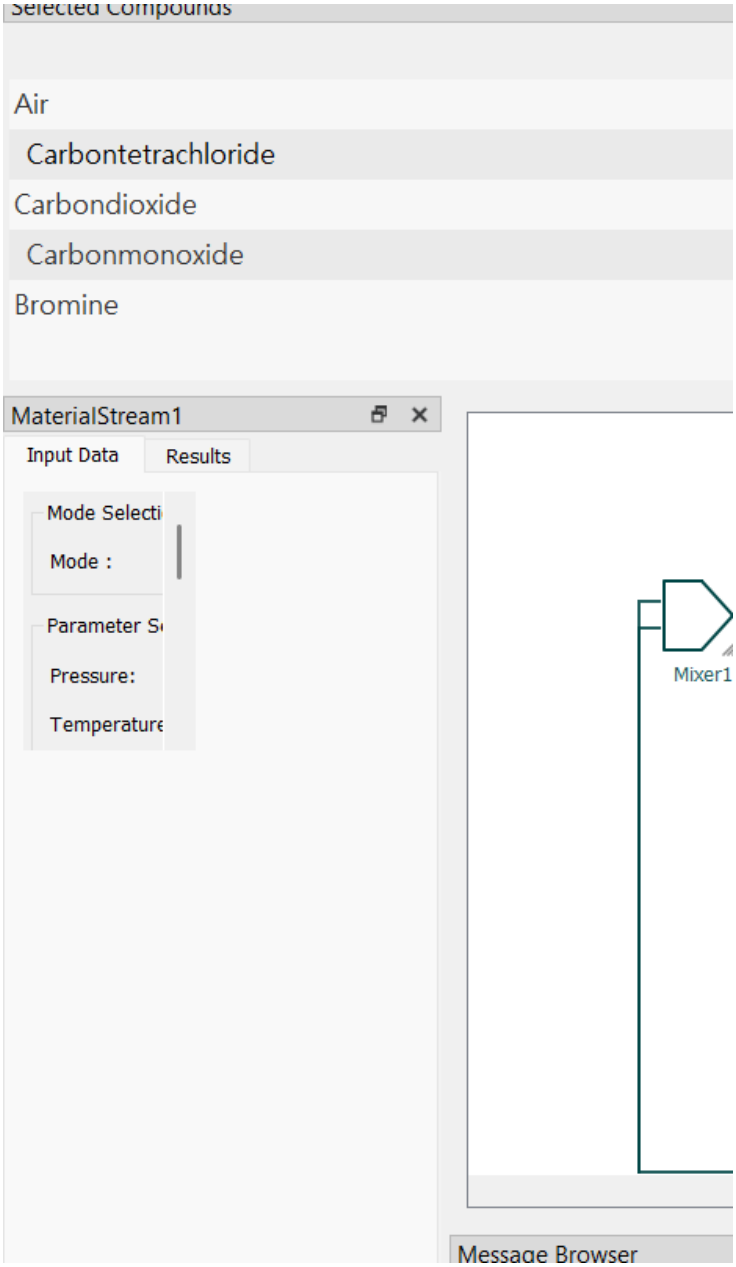

Figure 7.1: Legacy MaterialStream Dock Widget with Nested Scrollbar Clipping (Screenshot 2026-06-24 210136).

Inspection of DockWidgetMaterialStream.py revealed that input fields were encapsulated inside an inner fixed-size container within a nested QScrollArea. On standard displays, this forced vertical scrollbars to appear even for streams with only two parameters, clipping the critical 'Submit' button.

7.2 PR #93: Direct Layout and Dynamic Autosizing

In PR #93, I stripped out the nested QScrollArea completely. Input controls were moved directly into the main tab QVBoxLayout, configured with QLayout.SetDefaultConstraint and expanding size policies.

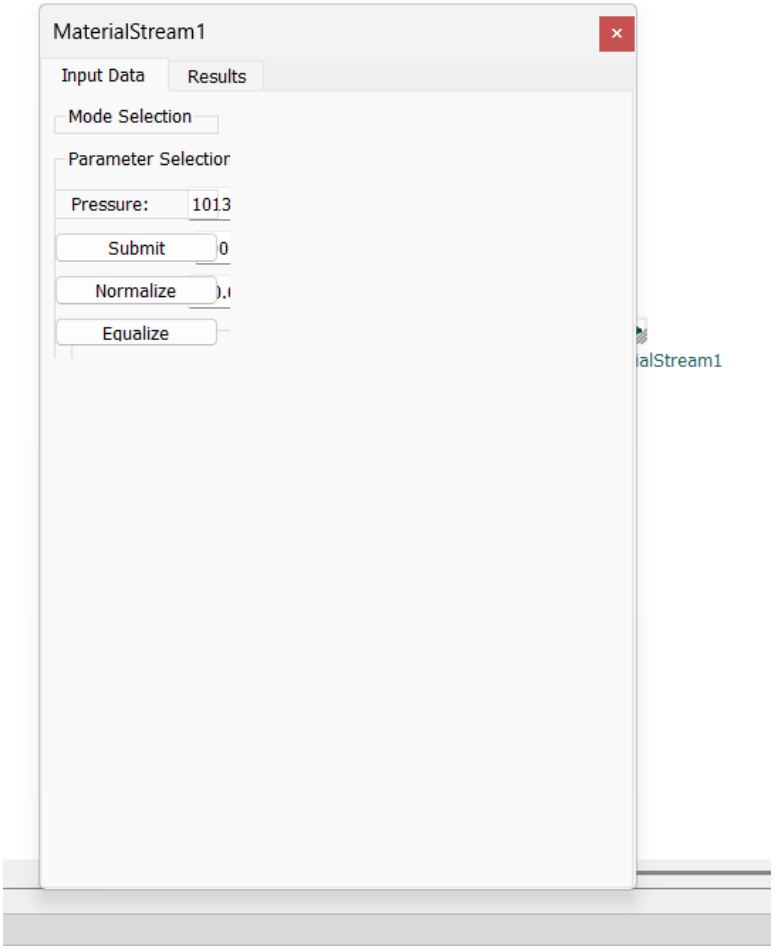

Figure 7.2: Layout Restructuring Removing Inner Scroll Area Container (Screenshot 2026-06-24 211444).

To guarantee that the dock widget adapts its physical dimensions to the number of active input fields, I implemented a dynamic _autosize() calculation tied to showEvent():

```
# --- Dynamic Autosizing in DockWidgetMaterialStream.py (PR #93) ---
def showEvent(self, event):
    super().showEvent(event)
    self._autosize()

def _autosize(self):
    """Dynamically resizes dock widget to fit content height without scrollbars."""
    hint = self.widget().sizeHint()
    screen_h = QApplication.desktop().screenGeometry().height()
    max_allowed_h = int(screen_h * 0.7)
    target_w = max(hint.width(), 240)
    target_h = min(hint.height(), max_allowed_h)
    self.resize(target_w, target_h)
```

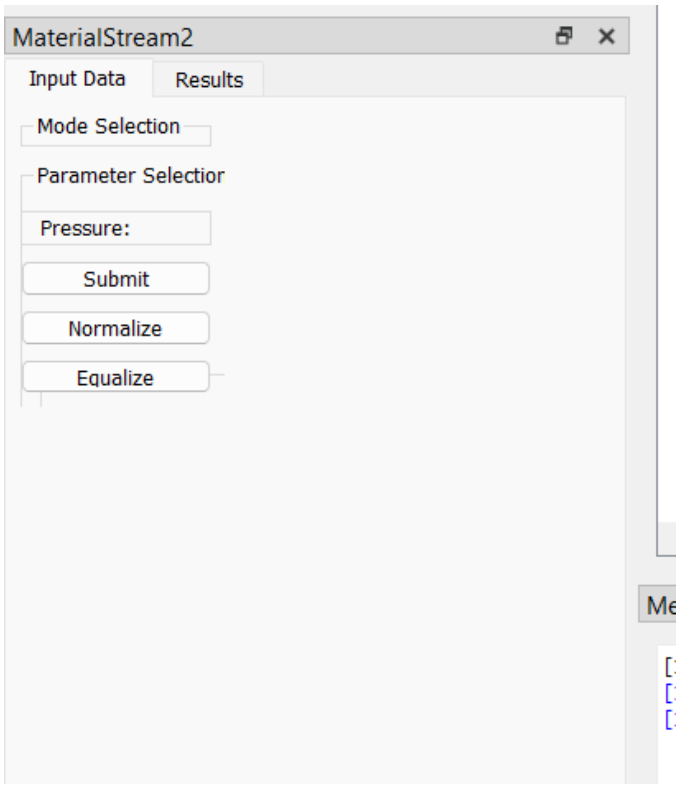

Figure 7.3: Dynamic Expansion of DockWidget Parameter Form Layout (Screenshot 2026-06-25 101507).

7.3 PR #95: Single-Pass Guard and Constraint Tuning

Real-world testing revealed that invoking _autosize() on every parameter edit triggered layout recalculation loops. I added an _has_autosized boolean guard so autosizing executes exactly once per dock presentation.

In Graphics.py, minimum dock widget constraints were relaxed from (280, 400) down to (220, 250), and maximum permitted widths in Qt Designer .ui files were increased to allow expansive multi-column tables.

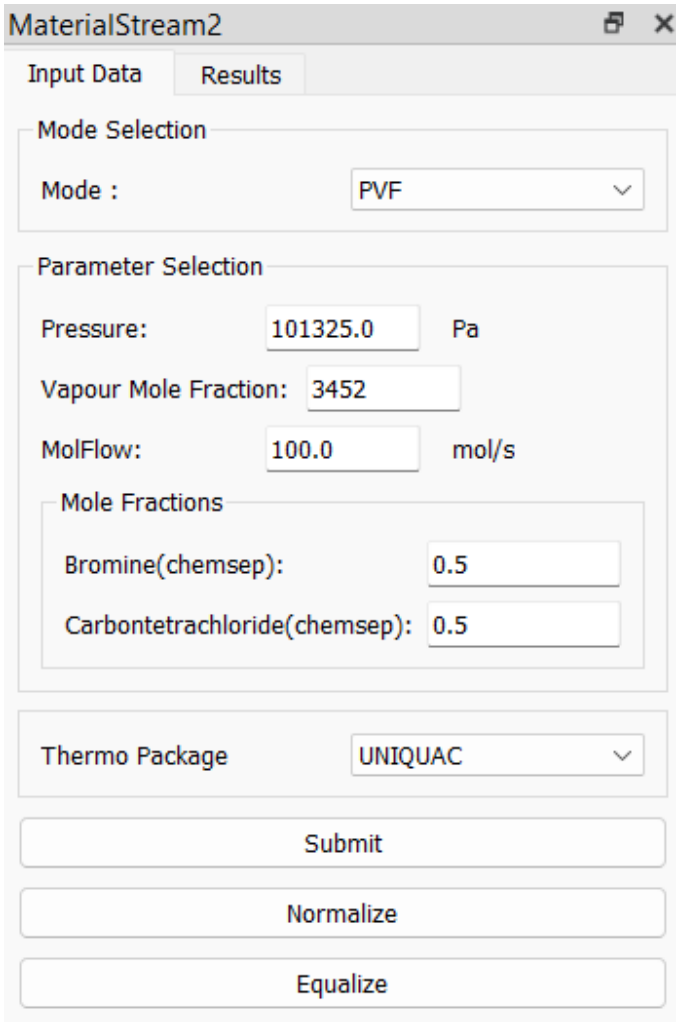

Figure 7.4: Fully Autosized Properties Panel Fitting All Controls Without Scrollbars (Screenshot 2026-06-25 103643).

7.4 VS Code-Style Draggable QSplitter Styling

To deliver smooth panel resizing, I authored custom CSS rules in mainApp.py for QSplitter handles:

```
/* --- VS Code-Style Draggable Splitters in mainApp.py (PR #95) --- */
QSplitter::handle:horizontal {
    background-color: #334155;
    width: 3px;
    margin: 0px;
}
QSplitter::handle:horizontal:hover {
    background-color: #38BDF8;
    width: 5px;
}
QSplitter::handle:horizontal:pressed {
    background-color: #0284C7;
}
```

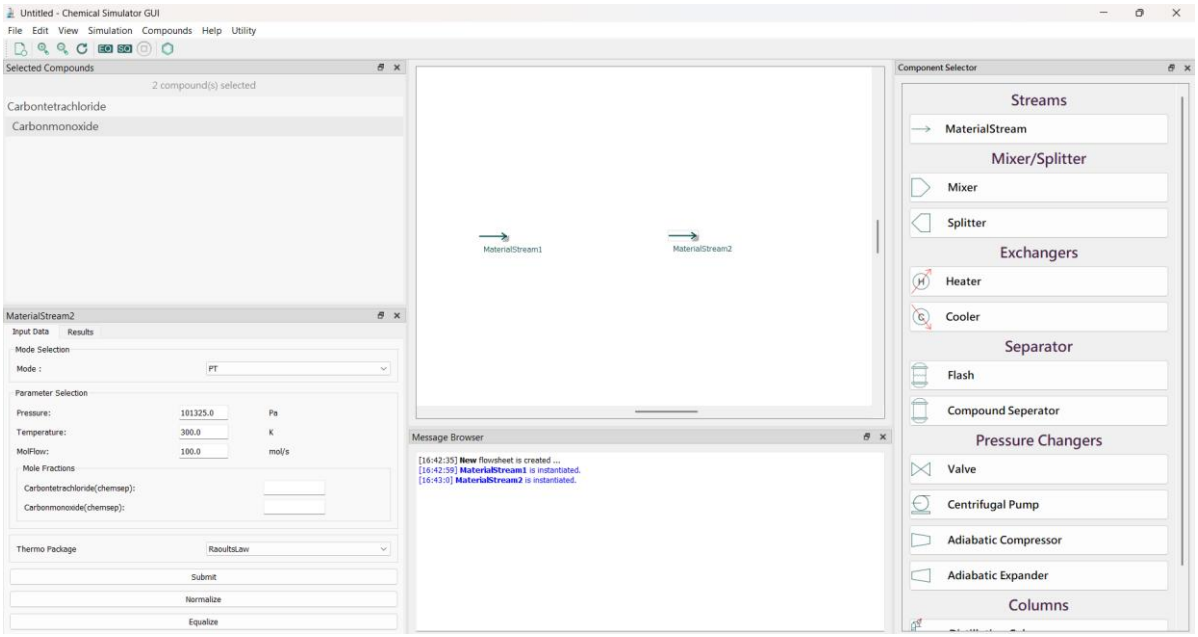

Figure 7.5: Full Application Window Showing Draggable VS Code-Style Panel Splitters (Screenshot 2026-06-25 164322).

7.5 Verification and Outcome

Combined, PR #93 (+63 / -43) and PR #95 (+23 / -9 across 4 files) were both merged on 25 June 2026, providing a fluid 3-panel workspace.

Chapter 8: Task 5 (PR #99) - About and Landing Page Redesign

PR #99 *merged Jul 3, 2026 | +335 / -147, 2 files | FOSSEE/Chemical-Simulator-GUI*

8.1 Problem Statement: Inconsistent Theme and Abrupt Transitions

The legacy About dialog used a light theme that clashed with the dark landing page, contained redundant close buttons, and opened abruptly without transitions. In addition, window controls on secondary dialogs required standard cross-platform behaviors.

8.2 Modern Slate-900 (#0f172a) Theme and Decoupled Callbacks

I redesigned About_page.py and Landing_page.py with a unified Slate-900 (#0f172a) dark theme, cyan accents (#38bdf8), and decoupled on_close callback dispatch:

```
# --- Modern Dark Theme and Callbacks in About_page.py (PR #99) ---
class AboutPage(QDialog):
    def __init__(self, on_close=None, parent=None):
        super().__init__(parent)
        self.on_close_callback = on_close
        self.setStyleSheet("""
            QDialog { background-color: #0F172A; color: #F8FAFC; border: 1px solid #334155;
}
            QLabel { color: #CBD5E1; font-family: 'Segoe UI', Arial; }
            QPushButton { background-color: #1E293B; color: #38BDF8; border: 1px solid
#0EA5E9; border-radius: 4px; padding: 6px 16px; }
            QPushButton:hover { background-color: #0284C7; color: #FFFFFF; }
        """)
```

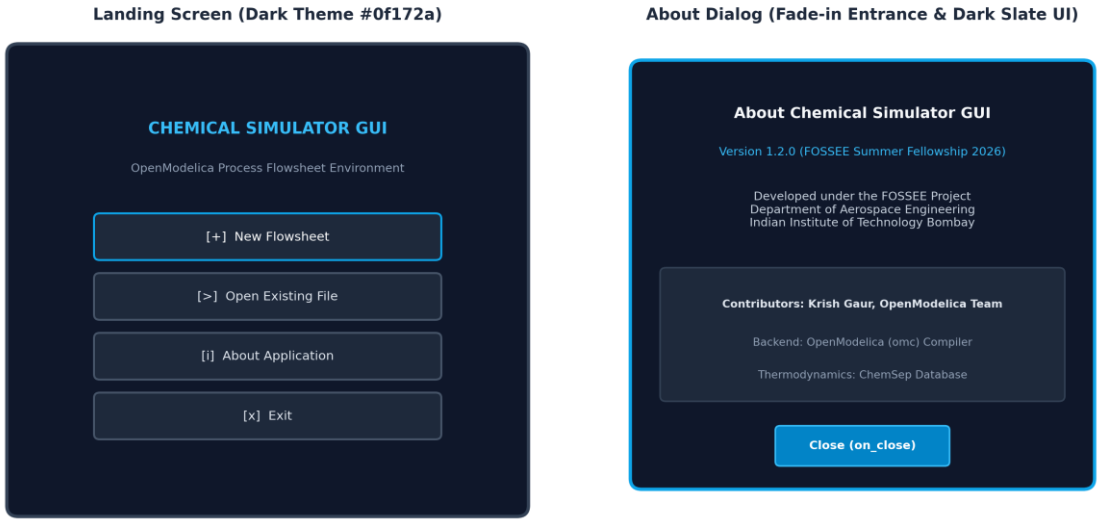

Figure 8.1: Modern Dark-Themed Landing Screen (left) and Animated About Dialog (right) (PR #99).

8.3 Opacity Fade-In Entrance Animation

To eliminate jarring modal popups, I integrated a smooth QPropertyAnimation operating on a QGraphicsOpacityEffect across a 300ms cubic easing curve:

```
# --- Fade-In Transition Effect ---
self.opacity_effect = QGraphicsOpacityEffect(self)
self.setGraphicsEffect(self.opacity_effect)
self.anim = QPropertyAnimation(self.opacity_effect, b'opacity')
self.anim.setDuration(300)
self.anim.setStartValue(0.0)
self.anim.setEndValue(1.0)
self.anim.setEasingCurve(QEasingCurve.OutCubic)
self.anim.start()
```

8.4 Memory-Safe Window Lifecycles and Outcome

I resolved critical C++ object deletion crashes by stashing and hiding MainApp instances rather than destroying them asynchronously. PR #99 merged on 3 July 2026 (+335 / -147 across 2 files), representing the largest single diff of my fellowship.

Chapter 9: Task 6 (PR #103) - Flowsheet Lifecycle and File Operations

PR #103 *merged Jul 14, 2026 | +117 / -34, 1 files | FOSSEE/Chemical-Simulator-GUI*

9.1 Problem Statement: Workspace Residue and Naming Collisions

Selecting 'New File' in the legacy codebase failed to execute a clean reset: active dock widgets remained floating, history stacks retained obsolete snapshots, and per-type naming counters persisted in memory. As a result, adding a mixer in a fresh project produced Mixer_6 instead of Mixer_1. Furthermore, unsaved changes were vulnerable to accidental discard without explicit user confirmation.

9.2 Confirmation Dialog Workflow on User Click

I implemented a modal prompt distinguishing automated startup initialization from explicit user clicks:

```
# --- Explicit 'New File' Confirmation Prompt in mainApp.py (PR #103) ---
def new_file(self, explicit_user_action=True):
    """
    Prompts user to save unsaved modifications only when explicitly triggered,
    then performs a deterministic clean-slate workspace reset.
    """
    if explicit_user_action and self.is_canvas_modified():
        reply = QMessageBox.question(
            self, 'Save Flowsheet?',
            'Do you want to save changes to the current flowsheet before creating a new
one?',
            QMessageBox.Save | QMessageBox.Discard | QMessageBox.Cancel,
            QMessageBox.Save
        )
        if reply == QMessageBox.Save:
            if not self.save_file():
                return  # Abort if save was canceled
        elif reply == QMessageBox.Cancel:
            return  # Abort new file action

    self._execute_deterministic_reset()
```

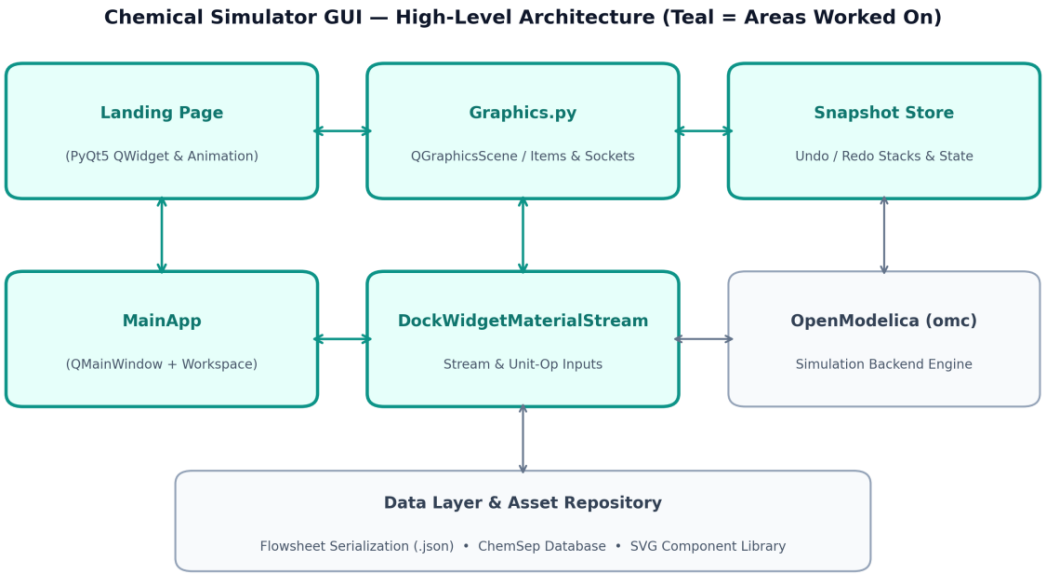

Figure 9.1: Deterministic New File Reset and Flowsheet JSON Serialization Pipeline (PR #103).

9.3 Deterministic Scene Reset Implementation

The _execute_deterministic_reset() routine enforces a clean state:

- **1. Canvas Teardown:** Clears all QGraphicsItems, connection lines, and selection bounding boxes.
- **2. Dock Cleanup:** Closes and destroys all floating and docked equipment property windows.
- **3. Tag Reset:** Resets per-type instance counters (self.name_counters = {}) so naming restarts at 1.
- **4. History Reset:** Re-initializes undo and redo stacks with a single pristine baseline snapshot.

9.4 Robust Flowsheet JSON Serialization

Save and load routines were hardened to guarantee complete round-trip preservation of canvas items, stream connections, and ChemSep thermodynamic models.

9.5 Verification and Outcome

PR #103 merged on 14 July 2026 (+117 / -34 lines in mainApp.py), concluding my fellowship contribution series.

Chapter 10: Contribution Statistics and Quantitative Analysis

10.1 Pull Request Timeline and Velocity

Across the fellowship, all seven contributions were delivered through feature-branch pull requests and merged into upstream master by mentor Chirag Koyande.

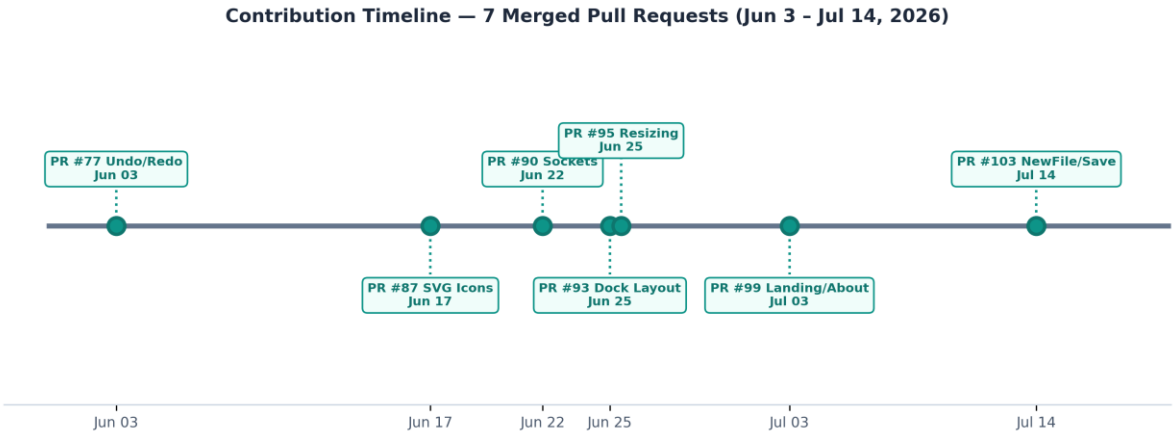

Figure 10.1: Timeline of Seven Merged Pull Requests Spanning June 3 to July 14, 2026.

10.2 LOC Metrics: Insertions and Deletions

In total, my pull requests contributed +929 insertions and -379 deletions across 31 files.

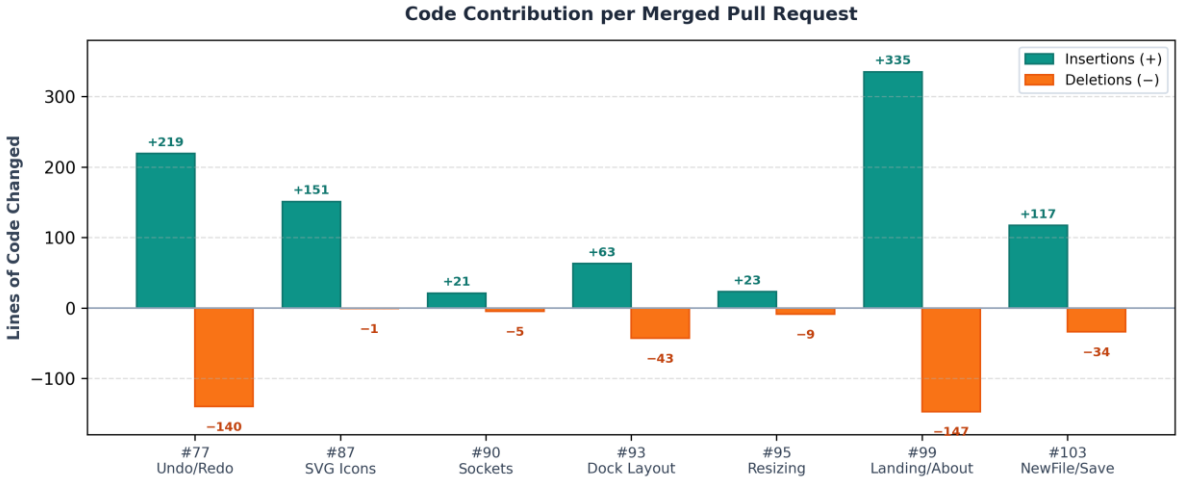

Figure 10.2: Code Contribution Metrics: Insertions (+) and Deletions (-) per Merged Pull Request.

10.3 Functional Effort Distribution

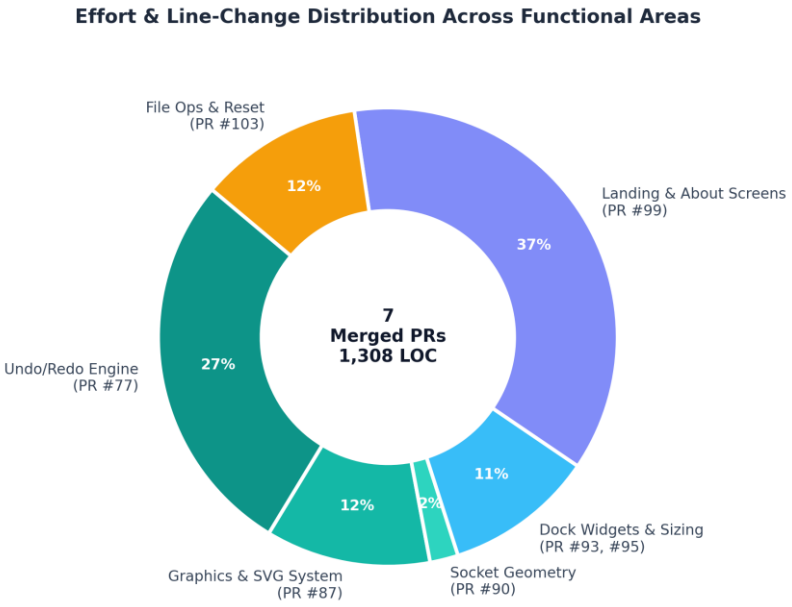

Figure 10.3: Functional Effort Distribution Across Core Architectural Subsystems.

10.4 Consolidated Contribution Table

Table 10.1: Consolidated Contribution Table of Seven Merged Pull Requests

PR #	Merged Date	Pull Request Title	Diff (Lines)	Files
#77	Jun 3, 2026	fix: Refactor undo/redo functionality and clean up related code	+219 / -140	7
#87	Jun 17, 2026	fix: Enhance graphics rendering and add SVG icons for various components	+151 / -1	15
#90	Jun 22, 2026	Refactor socket repositioning logic for relative positioning on resize	+21 / -5	1
#93	Jun 25, 2026	Refactor layout and resizing behavior for Input Data and DockWidget	+63 / -43	1
#95	Jun 25, 2026	Refactor DockWidgetMaterialStream and MainApp for	+23 / -9	4

		improved resizing		
#99	Jul 3, 2026	fix: Refactor AboutPage and LandingPage	**+335 / -147**	2
#103	Jul 14, 2026	Refactor MainApp to enhance 'New File' and improve save/load operations	**+117 / -34**	1

Chapter 11: Tools, Technologies and Development Workflow

11.1 Full Technology Stack Breakdown

The Chemical Simulator GUI ecosystem leverages modern Python desktop libraries, vector rendering engines, and numerical equation solvers. Table 11.1 details the software stack.

Table 11.1: Technical Stack and Software Dependency Matrix

Layer / Category	Technology / Library	Role in Chemical Simulator Architecture
Core Language	`Python 3.11`	Application logic, state serialization, and simulation scripting
GUI Framework	`PyQt5 (v5.15.10)`	QMainWindow, QDockWidgets, signals/slots, and event dispatch
2D Graphics Engine	`QGraphicsScene and QPainter`	Flowsheet canvas, bezier stream connections, and render hints
Vector Rendering	`QSvgRenderer and SVG`	Scalable unit-operation icon rendering with in-memory caching
Simulation Backend	`OpenModelica (omc)`	Sequential and equation-based flowsheet simulation compiler
Thermodynamic DB	`ChemSep / CAPE-OPEN`	Thermophysical compound properties database for stream inputs
Data Processing	`Pandas and NumPy`	Simulation result parsing, stream tables, and CSV exports
Result Plotting	`pyqtgraph`	In-application thermodynamic property and profile visualization
Version Control	`Git and GitHub`	Feature branching (fix/undoredo), code reviews, and rebase merges
Packaging	`PyInstaller (Spec)`	Windows single-bundle executable deployment specification

11.2 Feature-Branch Git Workflow and Rebase Discipline

All development followed a disciplined Git workflow: every pull request originated from fix/undoredo, was rebased onto upstream/master, underwent local verification on Windows and Linux, and was submitted with structured descriptions detailing the problem, solution, and impacted files.

11.3 Cross-Platform Linux and Windows QA

All UI modifications, dialog fade animations, and layout splitters were verified across both Windows 11 and Ubuntu Linux environments to ensure platform-independent rendering.

Chapter 12: Challenges Faced and Lessons Learned

12.1 Technical Challenges

- **Distributed State Invariants:** In graphical applications, deep-copying scene objects without aliasing live Qt pointers is difficult. I resolved this by isolating serializable dictionaries (serialize()) and building pure reconstruction routines.
- **Faithful Scene Restoration:** Restoring a flowsheet requires not only placing items, but re-wiring socket-to-socket connections and synchronizing per-class naming counters. Solving this prevented subtle tag collision bugs.
- **Coordinate Geometry:** Fixing socket drift required understanding Qt's item-local coordinate transformations versus scene coordinates, proving that fractional ratio invariants (fx, fy) are essential during resizing.
- **Layout Under Variability:** Removing nested scroll areas and implementing single-execution autosizing guards resolved layout thrashing across high-DPI displays.
- **Version Control Hygiene:** Resolving modify/delete conflicts on committed binary pickle files reinforced the importance of git repository hygiene.

12.2 Professional and Open-Source Engineering Insights

- **Review-Driven Development:** Submitting seven PRs reviewed by Chirag Koyande taught me that clear PR documentation, rationale, and reproduction steps are as critical as the code itself.
- **Upstream Discipline:** Maintaining a single clean feature branch over six weeks of upstream movement required constant synchronization.
- **High-Leverage Polish:** Addressing subtle bugs (socket drift, dock scrollbars, clean transitions) produced a transformative impact on perceived application quality.

Chapter 13: Conclusions and Future Scope

13.1 Summary of Contributions

Over seven merged pull requests on the Chemical Simulator GUI, the editing experience evolved from ad-hoc behaviors into a coherent, production-grade desktop platform:

- **1.** PR #77: Rebuilt undo/redo around centralized atomic snapshots with clean git history.
- **2.** PR #87: Authored 13 vector SVG icons and built a cached QSvgRenderer pipeline with render hints.
- **3.** PR #90: Eliminated connection socket drift during resize using fractional geometric invariants.
- **4.** PR #93 and #95: Modernized DockWidgetMaterialStream with dynamic autosizing and draggable splitters.
- **5.** PR #99: Integrated modern dark UI styling (#0f172a) and smooth entrance animations.
- **6.** PR #103: Hardened the 'New File' workflow and save/load serialization pipeline.

13.2 Skills Developed

- **Technical Mastery:** Qt Graphics View architecture, custom QGraphicsItem paint pipelines, QSvgRenderer caching, and fractional coordinate math.
- **Open-Source Engineering:** Git branching, conflict resolution, peer review defense, and open-source collaboration.

13.3 Future Roadmap for Chemical Simulator GUI

- **SVG Palette Expansion:** Complete the vector migration for any remaining secondary dialog icons.
- **Rotation Invariants:** Extend fractional coordinate invariants to handle 90-degree and 180-degree component rotations.
- **Live Simulation Diagnostics:** Surface OpenModelica solver logs and equation convergence metrics in a dedicated dock.

Appendix A: Complete Contribution Index and PR Links

All seven pull requests were merged into FOSSEE/Chemical-Simulator-GUI (branch master) from the author's fork (KrishGaur1354/Chemical-Simulator-GUI):

- **PR #77:** https://github.com/FOSSEE/Chemical-Simulator-GUI/pull/77 (Merged Jun 3, 2026)
- **PR #87:** https://github.com/FOSSEE/Chemical-Simulator-GUI/pull/87 (Merged Jun 17, 2026)
- **PR #90:** https://github.com/FOSSEE/Chemical-Simulator-GUI/pull/90 (Merged Jun 22, 2026)
- **PR #93:** https://github.com/FOSSEE/Chemical-Simulator-GUI/pull/93 (Merged Jun 25, 2026)
- **PR #95:** https://github.com/FOSSEE/Chemical-Simulator-GUI/pull/95 (Merged Jun 25, 2026)
- **PR #99:** https://github.com/FOSSEE/Chemical-Simulator-GUI/pull/99 (Merged Jul 3, 2026)
- **PR #103:** https://github.com/FOSSEE/Chemical-Simulator-GUI/pull/103 (Merged Jul 14, 2026)

Appendix B: Development Setup and Repository Map

Environment Reproduction Steps

```
:: 1. Clone repository
git clone https://github.com/FOSSEE/Chemical-Simulator-GUI.git
cd Chemical-Simulator-GUI

:: 2. Create virtual environment & install requirements
python -m venv env
env\\Scripts\\activate
pip install -r requirements-windows.txt

:: 3. Launch application
python src\\main\\python\\Landing_page.py
```

Repository Architecture Map

Table B.1: Repository Architecture Map and Subsystem Responsibilities

Repository Path	Architectural Responsibility
`src/main/python/mainApp.py`	Main application window, undo/redo orchestration, toolbar actions, New File / Save / Load
`src/main/python/utils/Graphics.py`	QGraphicsScene flowsheet canvas, socket repositioning, snapshot capture, SVG renderer cache
`src/main/python/DockWidgets/`	Parameter editors, layout autosizing, and dynamic control scaling
`src/main/python/Landing_page.py`	Dark-themed landing window, button routing, and MainApp lifecycle management
`src/main/python/About_page.py`	Modernized dark About dialog with opacity fade-in animation
`src/main/resources/base/icons/svg/`	Thirteen custom vector SVG chemical engineering component icons

References

- **1. FOSSEE Project, IIT Bombay:** https://fossee.in
- **2. FOSSEE Summer Fellowship 2026 Results:** https://fossee.in/fellowship/2026/results
- **3. Chemical Simulator GUI GitHub Repository:** https://github.com/FOSSEE/Chemical-Simulator-GUI
- **4. OpenModelica Compiler & Modeling Environment:** https://openmodelica.org
- **5. Qt 5 Graphics View Framework & QSvgRenderer Documentation:** https://doc.qt.io/qt-5/graphicsview.html
- **6. ChemSep / CAPE-OPEN Thermophysical Property Databases:** https://www.chemsep.org
- **7. PyQt5 Reference & Python Bindings Guide:** https://www.riverbankcomputing.com/software/pyqt/