FOSSEE SUMMER FELLOWSHIP REPORT

On development of

Chemical GUI Simulator

Submitted by

KRISH VINOD JOSHI

3rd Year B.Tech Student,CSE

VIT BHOPAL

Under the Guidance of

Prof. Prabhu Ramachandran

Department of Aerospace Engineering

Indian Institute of Technology Bombay

Mentors:

Chirag Koyande

August 2026

Acknowledgments

I would like to express my sincere gratitude to the FOSSEE team at IIT Bombay for the opportunity to participate in the Summer Fellowship program. This experience has been intellectually rewarding, allowing me to contribute to a significant technical open-source project alongside a dedicated team.

As a contributor to the Chemical Simulator GUI, I worked on implementing advanced features including drag-and-drop mechanics, complex user interface refactoring with PyQt, robust state management, and critical memory stability fixes. Beyond frontend development, I managed repository interactions, debugged deep framework-level segmentation faults, and implemented seamless undo/redo mechanics. These responsibilities provided me with a holistic understanding of software architecture and the lifecycle of open-source software.

I am deeply grateful to Prof. Prabhu Ramachandran for his leadership and vision in promoting technical excellence through the FOSSEE initiative.

This fellowship has been a defining chapter in my journey, equipping me with the skills, precision, and clarity for a professional career in computer science engineering.

Contents

Chapter 1

Introduction

1.1 FOSSEE Project

The FOSSEE (Free/Libre and Open Source Software for Education) project promotes the use of FLOSS tools in academia and research. It is part of the National Mission on Education through Information and Communication Technology (NMEICT), Ministry of Education (MoE), Government of India. The primary objective is to improve the quality of education and research by developing and deploying open-source software tools that can substitute expensive proprietary software.

1.1.1 Projects and Activities

The FOSSEE Project supports the use of various FLOSS tools to enhance education and research. Key activities include:

- **Textbook Companion:** Porting solved examples from standard textbooks using FLOSS.
- **Lab Migration:** Facilitating the migration of proprietary educational labs to FLOSS alternatives.
- **Chemical Process Simulators:** Developing intuitive interfaces for powerful chemical engineering simulation engines (like OpenModelica and DWSIM).

1.2 Chemical Simulator GUI

1.2.1 Overview and Purpose

Chemical Simulator GUI is a cross-platform software system developed under the FOSSEE project for creating, editing, and managing Process Flow Diagrams (PFDs) for

chemical engineering. The tool interfaces with the OpenModelica engine, providing a professional-grade diagramming environment and thermodynamic simulation capabilities.

1.2.2 System Architecture

The application follows a desktop-first architecture utilizing Python and PyQt5. This design ensures native performance across Windows, Linux, and macOS platforms. The frontend manages the flowsheet topology, interactive canvas, and property configurations, while the backend generates Modelica code (`Flowsheet.mo`) and invokes the OpenModelica Compiler (OMC) to simulate steady-state unit operations.

1.2.3 Evolution & Scope of Work

The focus of this fellowship period was to systematically overhaul the core graphical user interface, introduce interactive drag-and-drop mechanics, resolve deep-rooted memory leaks and C++ object destruction crashes, and implement complex mathematical normalization for thermodynamic streams.

Chapter 2

Task 1: Component Selector UI Redesign (PR #102)

2.1 Problem Statement

The original Component Selector dock widget occupied a significant portion of the screen real estate, reducing the available flowsheet canvas space. Users lacked the ability to collapse the menu intuitively, and the application's startup sizing was constrained by static Qt Layout overrides, leading to a cramped user experience on smaller displays.

2.2 Implementation Details

The component selector was completely refactored from a static XML-based UI configuration (`.ui` file) to a dynamic, programmatic layout managed directly in Python.

2.2.1 Collapse/Expand Toggle Mechanism

An arrow bar toggle was introduced to transition the panel between an icon-only strip (48px wide) and a fully expanded textual list (240px wide). To achieve seamless transitions without visual artifacts, pre-compiled CSS stylesheets were injected instantly during the toggle event.

```
def _toggle_component_panel(self):
    if not self._panel_collapsed:
        # Collapse State
        for btn, _ in self._comp_btn_data:
            btn.setText('')
            btn.setStyleSheet(self._btn_style_collapsed)
        for h in self._comp_headers:
            h.hide()
        self.dockWidget.setFixedWidth(48)
```

```
        self._panel_collapsed = True
    else:
        # Expand State
        for btn, text in self._comp_btn_data:
            btn.setText(f'     {text}')
            btn.setStyleSheet(self._btn_style_expanded)
        for h in self._comp_headers:
            h.show()
        self.dockWidget.setFixedWidth(240)
        self._panel_collapsed = False
```

Listing 2.1: Dynamic Stylesheet Application for Dock Toggle

2.2.2 Canvas Layout Constraints

To ensure the flowsheet canvas dynamically reclaims the freed space when the dock collapses, the centralwidget size policy in main.ui was updated from Maximum to Expanding. Furthermore, a startup sequence bug was resolved by applying setFixedWidth strictly after addDockWidget() is called, successfully bypassing Qt's internal layout engine overrides.

2.3 Outcome

The redesigned component selector offers a modern, compact, and responsive user experience, significantly increasing the workable canvas area while maintaining quick access to all unit operations.

Chapter 3

Task 2: Compound Selection State Management (PR #98)

3.1 Problem Statement

A critical bug was identified in the Compound Selector window: when a user opened the selector, made modifications to the compound checkboxes, and then clicked 'Cancel', reopening the selector resulted in a completely blank table. The previous implementation destructively wiped the global compound list and emptied the UI table regardless of the rejection state.

3.2 Implementation Details

3.2.1 Non-Destructive Cancellation

The cancel() method was updated to strictly dismiss the dialog using self.reject(), preventing the accidental erasure of the underlying data structure.

```
def cancel(self):
    # Removed: compound_selected.clear()
    # Removed: self.tableWidget.setRowCount(0)
    self.reject()
```

Listing 3.1: Safe Rejection Logic

3.2.2 State Synchronization on Visibility

To ensure the UI accurately reflects the unmodified global state upon reopening, a showEvent override was implemented. This hook automatically triggers a state synchronization routine every time the window is requested by the window manager, effectively discarding any aborted edits from previous sessions.

```
def sync_state(self):
    current_names = [n.replace('(chemsep)', '') for n in
   compound_selected]
    self.set_compounds(current_names)

def showEvent(self, event):
    super().showEvent(event)
    if hasattr(self, 'compounds_data') and self.compounds_data:
        self.sync_state()
```

Listing 3.2: UI Reversion via showEvent Hook

3.2.3 Deselection Support

The accept() routine was refactored to fully replace the global compound array with the new selections, properly enabling compound deselection—a feature that was previously broken.

3.3 Outcome

The compound selection process is now highly robust, preventing data loss on cancellation and correctly restoring visual states, leading to a much smoother simulation configuration workflow.

Chapter 4

Task 3: Process Lifecycle & Stability Fixes (PR #89)

4.1 Problem Statement

The application suffered from severe stability issues during workspace transitions (e.g., returning from the flowsheet to the main landing page). Users frequently encountered 'Not Responding' freezes, abrupt Segmentation Faults, corrupted mouse cursors, and Unicode encoding crashes on Windows consoles.

4.2 Identified Root Causes

1. **C++ Object Destruction Race Conditions:** Closing the flowsheet triggered deleteLater() on the MainApp instance. Because PyQt wraps underlying C++ QObjects, asynchronous destruction raced against the Qt event loop, causing the application to query freed memory spaces.

2. **Cursor Stack Pollution:** Override cursors (WaitCursor) set during simulations were not drained from the global QApplication stack, locking mouse interactions upon window closure.

3. **Console Encoding:** The use of the Unicode right-arrow character in debugging prints crashed the application on Windows systems running standard CP1252 code pages.

4.3 Implementation Details

4.3.1 Safe Window Stashing

Instead of attempting to aggressively garbage collect the complex MainApp window, a safe stashing mechanism was implemented. The old window is hidden and appended to a module-level list, preserving its memory block and completely eliminating C++ destruction race conditions.

```
_stashed_windows = []

def _cleanup_main_window(self):
    if self.main_window is not None:
        try:
            self.main_window.hide()
            # Removed: self.main_window.deleteLater()
        except Exception:
            pass
        _stashed_windows.append(self.main_window)
        self.main_window = None
```

Listing 4.1: Safe Window Stashing Mechanism

4.3.2 Robust Thread and Cursor Cleanup

The closeEvent hook was fortified to forcibly terminate active OpenModelica threads and drain the cursor stack before transitioning back to the landing page.

```
def restore_landing_page(self):
    # Drain the entire override cursor stack left by MainApp
    while QApplication.overrideCursor() is not None:
        QApplication.restoreOverrideCursor()

    self._cleanup_main_window()
    self.show()
    self.activateWindow()
```

Listing 4.2: Cursor Stack Draining

4.4 Outcome

The software now successfully transitions between workspaces without freezing or crashing, providing a stable, production-ready environment across all operating systems.

Chapter 5

Task 4: Material Stream Normalization (PR #76)

5.1 Problem Statement

When configuring material streams with numerous chemical compounds, users manually calculated mole fractions to ensure they summed precisely to 1.0. This process was prone to human error, resulting in Modelica solver failures. Additionally, the Input Data panel clipped components when the compound count exceeded five.

5.2 Implementation Details

5.2.1 Scrollable Layout Hierarchy

The Input Data panel was rebuilt using a QScrollArea hierarchy to dynamically handle large compound lists (up to 50+ elements) without UI clipping.

```
self.scrollArea = QScrollArea()
self.scrollArea.setWidgetResizable(True)
self.scrollArea.setHorizontalScrollBarPolicy(Qt.ScrollBarAlwaysOff)
self.scrollArea.setVerticalScrollBarPolicy(Qt.ScrollBarAsNeeded)

self.scrollWidget = QWidget()
self.innerLayout = QVBoxLayout(self.scrollWidget)

# Migrating legacy widgets to new layout
for w in widgets_to_move:
    w.setParent(None)
    self.innerLayout.addWidget(w)
```

Listing 5.1: Dynamic Scroll Area Injection

5.2.2 Equalize and Normalize Algorithms

Two rapid-configuration functions were implemented:

- **Equalize Logic:** Distributes fractions equally ($value = 1.0/count$).
- **Normalize Logic:** Scales inputs proportionally ($normalized = current/total$).

To circumvent floating-point precision errors that plague solver engines, the algorithms force the final element via $1.0 - \sum(previous)$.

```
def normalize(self):
    values = [float(l.text()) if l.text().strip() else 0.0 for l in
   self.x_pclist]
    total = sum(values)

    if total > 0:
        sum_norm = 0
        for i in range(len(self.x_pclist) - 1):
            normalized_val = round(values[i] / total, 4)
            self.x_pclist[i].setText(str(normalized_val))
            sum_norm += normalized_val

        # Guarantee perfect unity
        self.x_pclist[-1].setText(str(round(1.0 - sum_norm, 4)))
```

Listing 5.2: Normalization Algorithm with Precision Guard

5.3 Outcome

Users can now effortlessly balance complex multicomponent streams, and the UI remains fully accessible regardless of the process complexity.

Chapter 6

Task 5: Drag-and-Drop & Deletion (PR #74)

6.1 Problem Statement

The application strictly relied on a 'click-to-add' mechanism that placed new unit operations blindly in the center of the viewport, often stacking them on top of one another. Furthermore, the environment lacked a robust mechanism for deleting components and their interconnected graphical lines.

6.2 Implementation Details

6.2.1 Drag-and-Drop Event Filtering

A custom QObject event filter was engineered to capture mouse events on the sidebar buttons, package the component metadata using QMimeData, and execute a native Qt drag operation.

```
class DragButtonFilter(QObject):
    def eventFilter(self, obj, event):
        if event.type() == QEvent.MouseMove:
            if event.buttons() & Qt.LeftButton and self.startPos:
                if (event.pos() - self.startPos).manhattanLength() >=
   QApplication.startDragDistance():
                    drag = QDrag(obj)
                    mimeData = QMimeData()
                    mimeData.setText(self.component_type)
                    drag.setMimeData(mimeData)

                    pixmap = obj.grab()
```

```
                    drag.setPixmap(pixmap.scaled(64, 64, Qt.
    KeepAspectRatio))
                    drag.exec_(Qt.CopyAction)
                    self.startPos = None
                    return True
        return False
```

Listing 6.1: Drag Execution Pipeline

6.2.2 Viewport Coordinate Mapping

Upon dropping the item, the DropFilter translates the raw screen coordinates into localized QGraphicsScene coordinates using mapToScene(), passing the precise vector to the instantiation engine.

6.2.3 Graph Cleanup and Deletion Routing

A comprehensive deletion algorithm was built to safely wipe components from existence. It targets the physical item, cleans the interconnected streams recursively, unbinds the associated dock widgets, and removes references from the global topology matrix.

```
# Collect all connected lines if we're deleting nodes
additional_lines = set()
for item in items_to_delete:
    if isinstance(item, NodeItem):
        for socket in (item.input + item.output):
            for line in (socket.in_lines + socket.out_lines):
                if line not in items_to_delete:
                    additional_lines.add(line)
items_to_delete.extend(list(additional_lines))
```

Listing 6.2: Recursive Connection Cleanup

6.3 Outcome

The flowsheet is now highly interactive. The intuitive drag-and-drop system, combined with robust deletion protocols, brings the interface's usability up to the standard of commercial chemical process simulators.

Chapter 7

Conclusions and Future Scope

7.1 Conclusion

The enhancements implemented throughout this FOSSEE fellowship drastically stabilized and modernized the Chemical Simulator GUI. By systematically tackling the core infrastructure issues—such as C++ lifecycle races, layout constraints, and event routing—the software is now significantly more robust. The introduction of Drag-and-Drop and smart thermodynamic stream normalization empowers chemical engineering students and researchers to build and simulate complex process diagrams with ease and accuracy.

7.2 Future Scope

Moving forward, the architecture laid down during this cycle can be expanded upon in several directions:

- **Undo/Redo Command Pattern:** Refactoring the snapshot-based undo system into a granular Command Pattern approach to optimize memory footprint during extensive sessions.
- **Dynamic Canvas Routing:** Implementing A* pathfinding algorithms for process streams to enable automatic orthogonal routing, preventing streams from overlapping unit operations.
- **Real-Time Compilation Feedback:** Integrating WebSocket or background worker hooks directly into the OMC log outputs to stream real-time compilation errors and warnings straight into the graphical workspace.

Chapter 8

Appendix A: Core Implementation Code

This appendix provides a high-level representation of the extensive codebase modifications applied to the PyQt frontend. Due to the vast size of the commits, carefully selected representative blocks from the PR diffs are displayed here to illustrate the technical complexity handled during the fellowship.

8.1 Component Selector & Collapse Toggle (PR #102)

```
def _build_component_selector(self):
    """Rebuild the Component Selector dock with collapse/expand toggle.
    """
    ACCENT       = '#0a84ff'
    BG           = '#ffffff'
    HEADER_BG    = '#f2f2f2'

    # Configure dockWidget
    self.dockWidget.setFeatures(
        QDockWidget.DockWidgetFloatable |
        QDockWidget.DockWidgetMovable |
        QDockWidget.DockWidgetClosable
    )
    self.dockWidget.setWindowTitle('Components')
    self.dockWidget.setFixedWidth(240)

    # Content widget
    content = QWidget()
    content.setStyleSheet(f'background: {BG};')
    main_layout = QVBoxLayout(content)
    main_layout.setContentsMargins(0, 0, 0, 0)

    # Toggle button
    toggle_btn = QPushButton('>')
```

```
    toggle_btn.setFixedHeight(24)
    toggle_btn.setCursor(Qt.PointingHandCursor)
    toggle_btn.clicked.connect(self._toggle_component_panel)
    self._panel_toggle_btn = toggle_btn
    main_layout.addWidget(toggle_btn)

    # Scroll area logic
    scroll = QScrollArea()
    scroll.setWidgetResizable(True)
    scroll.setHorizontalScrollBarPolicy(Qt.ScrollBarAlwaysOff)

    # Building buttons from data matrix
    for section_name, buttons in sections:
        header = QLabel(section_name.upper())
        self._comp_headers.append(header)
        for attr_name, display_text, icon_file in buttons:
            btn = QPushButton(f'     {display_text}')
            self._comp_btn_data.append((btn, display_text))
            scroll_layout.addWidget(btn)
```

8.2 Cursor Drain & Memory Stashing (PR #89)

```
def restore_landing_page(self):
    # Drain the entire override cursor stack left by MainApp/Graphics/
    Container
    # before showing the landing page, to prevent corrupted mouse
    events.
    while QApplication.overrideCursor() is not None:
        QApplication.restoreOverrideCursor()

    # Safely clean up old main window using Qt event loop
    self._cleanup_main_window()

    # Show landing page (windowed)
    self.show()

    # Make sure it is active
    self.activateWindow()
    self.raise_()

def _cleanup_main_window(self):
    if self.main_window is not None:
        # Clear module-level variables
        try:
            if _dock_widget_lst is not None: _dock_widget_lst.clear()
```

```
            if _node_lst is not None: _node_lst.clear()
        except Exception: pass

        try:
            self.main_window.hide()
        except Exception:
            pass

        # Stash preventing Python GC from triggering C++ deleteLater
        _stashed_windows.append(self.main_window)
        self.main_window = None
```

8.3 Stream Fraction Normalization (PR #76)

```
def equalize(self):
    try:
        noc = len(self.x_pclist)
        if noc > 0:
            val = 1.0 / noc
            sum_val = 0
            for i in range(noc - 1):
                v = round(val, 4)
                self.x_pclist[i].setText(str(v))
                sum_val += v
            self.x_pclist[-1].setText(str(round(1.0 - sum_val, 4)))
    except Exception as e:
        print(e)
```

8.4 Event Filter for Canvas Drop (PR #74)

```
class DropFilter(QObject):
    def __init__(self, parent, main_app):
        super().__init__(parent)
        self.main_app = main_app

    def eventFilter(self, obj, event):
        if event.type() == QEvent.DragEnter:
            if event.mimeData().hasText():
                event.acceptProposedAction()
                return True
        elif event.type() == QEvent.DragMove:
            if event.mimeData().hasText():
                event.acceptProposedAction()
```

```
                return True
        elif event.type() == QEvent.Drop:
            component_type = event.mimeData().text()
            # map from viewport coordinates to scene coordinates
            pos = self.main_app.graphicsView.mapToScene(event.pos())
            self.main_app.component(component_type, pos=pos)
            event.acceptProposedAction()
            return True
        return False
```

Chapter 9

Appendix B: Extended Database Generation

The integration required standardizing thermodynamic schema files for OpenModelica bindings.

```
within Simulator.Generated;
package data
  record GeneralProperties
      extends Modelica.Icons.Record;
    parameter Integer SN;
    parameter String name;
    parameter String CAS;
    parameter Real Tc (unit='K');
    parameter Real Pc (unit='Pa');
    parameter Real Vc (unit='m3/kmol');
  end GeneralProperties;

  record Argon
      extends Modelica.Icons.Record;
    extends GeneralProperties(SN = 2, name = 'Argon', CAS = '7440-37-1'
    , Tc = 150.86, Pc = 4898000, Vc = 0.07457, Cc = 0.291, Tb = 87.27,
    Tm = 83.8039, TT = 83.8, TP = 68906.1, MW = 39.948, LVB = 0.0291, AF
     = -0.002, SP = 14138.3, DM = 0, SH = 0.0, IGHF = 0, GEF = 0, AS =
    154732, HFMP = 1184900, HOC = 0, LiqDen = {105, 3.803, 0.286,
    150.86, 0.2984, 0}, VP = {101, 44.369, -1126.1, -4.5688,
    0.000062339, 2}, LiqCp = {16, 46085, -1304.5, 21.195, -0.015382,
    0.000033063}, HOV = {106, 7981000, 0.099752, 0.32009, -0.11898,
    0.031141}, VapCp = {16, 20786, 0, 0, 0, 0});
  end Argon;
end data;
```

— END OF REPORT —