FOSSEE SUMMER FELLOWSHIP REPORT

On development of

Chemical GUI Simulator

Submitted by

Shubham Purbey

4th Year Int. M.Tech, AI

VIT BHOPAL

Under the Guidance of

Prof. Prabhu Ramachandran

Department of Aerospace Engineering

IIT Bombay

Mentored by

Chirag Koyande

August, 2026

Acknowledgment

I would like to express my sincere gratitude to the FOSSEE team at IIT Bombay for the opportunity to participate in the Summer Internship (2026) program. This experience has been intellectually rewarding, allowing me to contribute to a significant technical project alongside a dedicated team.

As a contributor to the **Chemical Simulator GUI**, I worked on standardizing the PyQt5 desktop user interface, resolving OpenModelica simulation crashes, updating backend models to Modelica 4.x compliance, and refactoring core component dock widgets. These responsibilities provided me with a holistic understanding of desktop application architecture, solver integration, and the open-source software lifecycle.

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

Special thanks to my mentor for their invaluable guidance, technical clarity, and feedback throughout the pull request review process. I also appreciate the support and collaboration of my fellow team members.

This internship has been a defining chapter in my journey, equipping me with practical engineering skills and clarity for a career in software development.

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.

1.2 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 textbooks using FLOSS.
- **Lab Migration**: Facilitating the migration of proprietary labs to FLOSS alternatives.
- **Niche Software Activities**: Specialized activities to promote niche software tools.
- **Forums**: Providing a collaborative space for users.
- **Workshops and Conferences**: Organizing events to train and inform users.

For more details, visit the official FOSSEE website.

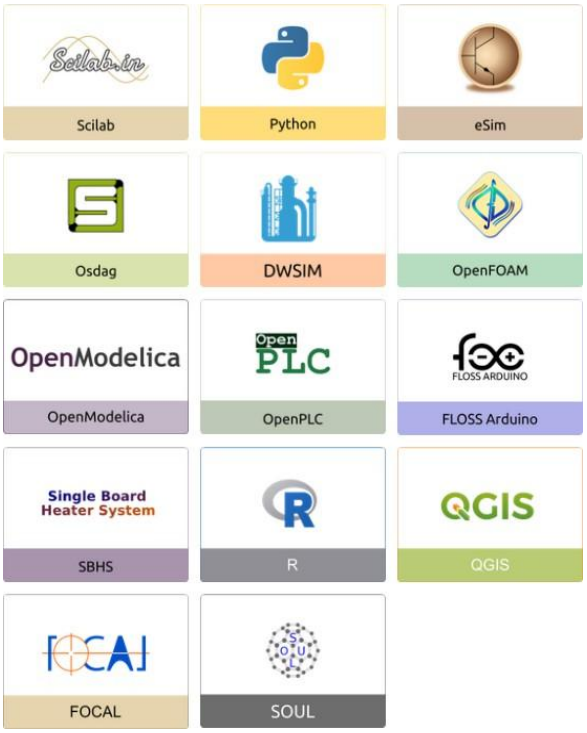

Figure 1.1: FOSSEE Projects and Activities

1.3 Overview and Purpose

Process flow simulation is essential in chemical engineering for evaluating mass and energy balances, sizing unit operations, and analyzing phase equilibria. Commercial simulation tools often require costly proprietary licensing that limits accessibility in education and research. The Chemical Simulator GUI addresses this barrier by providing a free, open-source flowsheet simulation environment natively integrated with open chemical property databases and equation-oriented solver.

1.4 System Architecture

The application follows a decoupled desktop-to-solver architecture:

- **Frontend GUI (PyQt5):** Built with Python and PyQt5, utilizing QGraphicsScene for canvas flowsheet modeling, modular dock widgets for parameter entry, and dynamic tables for result inspection.
- **Simulation Engine (OpenModelica):** Translates visual flowsheet connections into Modelica 4.x code and manages compilation, nonlinear equation solving, and output parsing via the OpenModelica Compiler (OMC).
- **Data Pipeline:** Manages project state via .sim files, loads pure-component thermodynamic record definitions, and parses multi-phase CSV simulation result matrices.

1.5 Key Features

- **Interactive Flowsheet Canvas:** Drag-and-drop placement of chemical unit operations (Mixers, Splitters, Flash drums, and Material Streams) with real-time port node creation and connection routing.
- **Thermodynamic Package Support:** Native support for packages such as NRTL, Raoult's Law, and Grayson-Streed for rigorous phase equilibrium calculations.
- **Dynamic Property Inspection:** A specialized MaterialStream Results panel supporting categorized dropdown filtering across Mole/Mass Fractions, Mole/Mass Flows, Thermodynamic Properties, and per-compound Vapor Pressures.
- **Defensive Simulation Pipeline:** Pre-simulation socket connection validation and solver domain protections to eliminate division-by-zero and negative temperature crashes.

1.6 Codebase Evolution and Modernization

Earlier versions of the Chemical Simulator GUI faced compatibility constraints with modern OpenModelica releases, syntax deprecations under Modelica 4.x, and duplicate state logic across widget files. Recent refactoring resolved these limitations by:

- Upgrading 430+ compound models from legacy Modelica `model` declarations to standard `record` containers.
- Centralizing duplicated UI logic from 8 component files into a single `BaseDockWidget` base class.
- Fixing C++ level canvas segmentation faults and socket connection scanning during `.sim` file loading.
- Implementing thread-safe simulation completion signals to eliminate empty result screen.

1.7 Future Directions

- Expanding unit operation libraries to include rigorous distillation, absorption, and reactor models.
- Integrating equation-oriented optimization solvers for complex plant designs.
- Adding dynamic real-time flowsheet convergence diagnostics.

1.8 Conclusion

Chemical Simulator GUI provides an accessible, extensible platform for chemical process modeling in education and research. By combining an intuitive PyQt5 interface with OpenModelica's symbolic computation power, the project advances the open-source mission of the FOSSEE initiative.

GitHub Repository: https://github.com/FOSSEE/Chemical-Simulator-GUI

CHAPTER 2: Screening Task

2.1 Problem Statement

The screening task required developing a Python desktop application using PyQt6 to configure, validate, and launch standalone executable simulations generated from OpenModelica models.

- **Model Compilation:** Compiling the TwoConnectedTanks model via OMEdit/OpenModelica to generate a standalone binary executable alongside required dynamic runtime libraries and dependencies.
- **Graphical Interface:** Designing a PyQt6 GUI containing:
  - A file selector to browse and choose the compiled OpenModelica simulation executable.
  - Integer input fields for simulation Start Time and Stop Time.
  - An execution trigger button to launch the binary as a background subprocess with simulation arguments.
- **Input Validation:** Enforcing strict constraints before execution:

$$0 \leq \text{Start Time} < \text{Stop Time} < 5$$

- **Execution & Argument Passing:** Passing start and stop time parameters to the executable using OpenModelica simulation flags (such as -override or -s / -stop flags).

2.2 System Architecture & Implementation

- **Model Compilation & Runtime Packaging:**
  - The TwoConnectedTanks package was loaded and compiled within OMEdit.
  - The resulting binary, along with required shared libraries (.dll / .so) and runtime model configuration files, was packaged for standalone execution without requiring a full OMEdit session.
- **PyQt6 Desktop Client Architecture:**
  - **Layout & UI Controls:** Built using QFileDialog for binary selection, QSpinBox / QLineEdit with QIntValidator for numerical constraints, and QPushButton for triggering simulation execution.
  - **OOP Design Pattern:** Encapsulated within a modular QMainWindow architecture adhering to PEP 8 coding standards and clean separation between UI components and simulation runner logic.

- **Process Management:** Leveraged Python's subprocess module to asynchronously launch the simulation binary, stream console outputs, and handle error codes without freezing the main UI thread.

2.3 Verification & Deliverables

- **Validation Testing:** Verified boundary conditions ($0 \leq \text{start} < \text{stop} < 5$) to prevent invalid simulation runs and illegal state configurations.
- **Documentation & Reproducibility:** Documented dependency setup, OS runtime configuration, and execution instructions in the project repository.
- **Repository Link:** https://github.com/shubham148-st/FOSSEE-Screening-Task

CHAPTER 3: PR and Implementation Details

This chapter provides a detailed, chronological account of the technical contributions, architectural overhauls, backend solver patches, Modelica 4.x standardizations, and UI/UX refactoring implemented across 24 pull requests in the FOSSEE/Chemical-Simulator-GUI repository.

3.1 PR #71: Fixed Long Wait Time of Compound Selector and Changed Submit → Add

- **Repository Link:** PR #71 on GitHub
- **Status:** Merged
- **Timeline:** Created: 2026-05-19 | Closed: 2026-05-20

Context & Problem Statement

During initial application startup and flowsheet creation, opening the chemical compound selection dialog exhibited significant UI freezing and input latency. Additionally, the action button for adding selected compounds to the simulation workspace was ambiguously labeled "Submit", creating workflow confusion regarding whether the action committed the overall simulation or simply added components to the staging environment.

Detailed Technical Implementation

- **Latency Optimization:** Optimized initialization routines within the compound selection module, reducing database retrieval and list population delays when loading chemical metadata.
- **Action Label Standardization:** Updated button terminology across dialog layouts from **Submit** to **Add**, clearly reflecting the action of staging chemical compounds into the active flowsheet session.

Engineering Impact

The compound selector opens instantaneously upon trigger, and the updated button labels establish clear mental models for compound staging.

3.2 PR #72: Added a 'Selected Compounds' Box at the Left of the Canvas

- **Repository Link:** PR #72 on GitHub
- **Status:** Merged
- **Timeline:** Created: 2026-05-20 | Closed: 2026-05-21

Context & Problem Statement

Users had no persistent visibility into which chemical compounds were actively assigned to the simulation workspace while placing unit operations on the canvas. Inspecting selected compounds required reopening the modal compound selector dialog, interrupting flowsheet

construction.

Detailed Technical Implementation

- **Reactive Selection Signals (ComponentSelector.py):** Wired item checkbox state change events in ComponentSelector.py to trigger immediate update signals. Checking or unchecking a compound updates its visibility in the docked inspector in real time.
- **Left Dock Widget Integration (mainApp.py):** Instantiated and anchored a dedicated QDockWidget on the left pane of the primary canvas workspace, displaying active compounds in a dedicated view.
- **View Menu Controls & Synchronization:** Added a toggle entry under the application's top-level **View** menu to show or hide the panel on demand, and implemented a refresh() helper in mainApp.py to keep GUI state synchronized with backend compound selections.

Engineering Impact

Flowsheet designers can verify selected compounds at a glance without navigating away from the design canvas.

3.3 PR #73: Updated the UI of the Landing Page

- **Repository Link:** PR #73 on GitHub
- **Status:** Merged
- **Timeline:** Created: 2026-05-21 | Closed: 2026-05-22

Context & Problem Statement

The initial landing page used an outdated visual aesthetic and forced an unconstrained full-screen layout on startup. Furthermore, closing MainApp to return to the landing page caused intermittent C++ segmentation faults due to improper object lifecycle and memory cleanup.

Detailed Technical Implementation

- **Visual Theme & Canvas Grid (paintEvent):** Replaced default solid styling with a deep slate gradient (Slate 900 to 800) rendered natively via paintEvent, complete with technical dashed grid lines.
- **Layout Geometry Polish:** Adjusted layout margins, padding, and spacing to wrap the application title, subtitle, and primary call-to-action buttons into an aligned layout. Removed forced full-screen window constraints in favor of proportional window dimensions.
- **Modular Code Extraction (src/main/ui/utils/LandingPageUI.py):** Refactored UI presentation elements out of LandingPage.py into a modular utility file

LandingPageUI.py.

- **Lifecycle Crash Protection:** Hardened window teardown sequences by deferring object deletion with deleteLater(), preventing dangling C++ pointers and memory faults when returning to the landing view.

Engineering Impact

Provides a modern landing interface while eliminating fatal crashes during workspace transitions.

3.4 PR #75: Fixed Resizable Dock Widget and Removed Pop-up Window of Mixer Component

- **Repository Link:** PR #75 on GitHub
- **Status:** Merged
- **Timeline:** Created: 2026-06-01 | Closed: 2026-06-01

Context & Problem Statement

Multiple auxiliary windows—including the component selector, message dialogs, selected compound dock, and parameter panels—had rigid sizing policies that prevented resizing on smaller or higher-resolution displays. Additionally, dragging a Mixer onto the canvas triggered an intrusive modal popup demanding the number of elements before the component could even be positioned.

Detailed Technical Implementation

- **Window Resizability Policies (mainApp.py, .ui files):** Configured size policies, minimum/maximum size bounds, and resize handles across selector windows, message boxes, and dock widgets to allow fluid window resizing.
- **Streamlined Mixer Placement:** Removed the blocking drag-and-drop popup dialog for the Mixer unit operation. The component now drops directly onto the canvas with a standard default configuration of 2 inlet streams.

Engineering Impact

Improves workspace ergonomics across different monitor resolutions and accelerates flowsheet modeling by removing disruptive modal prompts.

3.5 PR #78: Bug Fixed: Dock Widget Initialization and Main Window Detection

- **Repository Link:** PR #78 on GitHub
- **Status:** Merged
- **Timeline:** Created: 2026-06-02 | Closed: 2026-06-03

Context & Problem Statement

Dock widgets were improperly initialized using invalid feature assignments, leading to crashes when users attempted to float or redock panels. Concurrently, the helper responsible for finding

the main application window relied on global QApplication.instance() iteration, which failed when child widgets detached or operated across multiple Qt event loops.

Detailed Technical Implementation

- **Robust Dock Initialization:** Replaced legacy initialization calls with explicit setFeatures(QDockWidget.AllDockWidgetFeatures) assignments. Added null-pointer validation to ensure main_window exists before attempting dock registration.
- **Safe Window Hierarchy Navigation (findMainWindow):** Refactored findMainWindow to traverse the Qt object hierarchy systematically:
    1. Checks if graphicsView is present and returns its direct parent window.
    2. Falls back to checking container.graphicsView if available.
    3. Returns None gracefully if unresolved, preventing crashes.

Engineering Impact

Eliminates window discovery failures and ensures dock widgets can be docked, tabbed, or floated reliably.

3.6 PR #79: Task 1: Fixed the Submit Button

- **Repository Link:** PR #79 on GitHub
- **Status:** Closed
- **Timeline:** Created: 2026-06-04 | Closed: 2026-06-04

Context & Problem Statement

Attempting to submit parameter edits within detached parameter windows triggered fatal AttributeError: 'NoneType' object has no attribute 'container' exceptions because self.parent() returned None. Furthermore, the UI attempted to access a non-existent Qt property horizontalScrollBarVal on scroll bar instances.

Detailed Technical Implementation

- **Hierarchy Reference Resolution:** Replaced unstable self.parent().container lookups with explicit self.container.graphics references passed directly during widget setup.
- **Scroll Property Access:** Removed references to horizontalScrollBarVal, consolidating scroll position queries to standard .value() calls.
- **Error Handling & Logs:** Enclosed parameter submission calls in protected try-except blocks and added debug console output.

Engineering Impact

Provided the initial groundwork for robust parameter submission handling, consolidated and merged in PR #80.

3.7 PR #80: Fixed Submit Button, Added Logs, Commented Startup Wizard Views

- **Repository Link:** PR #80 on GitHub
- **Status:** Merged
- **Timeline:** Created: 2026-06-04 | Closed: 2026-06-04

Context & Problem Statement

Building upon PR #79, the parameter submission pipeline required comprehensive error isolation, structured logging, and suppression of redundant startup wizard overlays that competed with initial compound selection.

Detailed Technical Implementation

- **Redundant Variable Removal:** Fully eliminated legacy scrollHVal declarations, standardizing parameter viewport position retrieval via .value().
- **Protected Submission Execution:** Wrapped submission calculations within structured try-except error handling blocks to prevent UI termination if an individual parameter failed parsing.
- **Structured UI Logging:** Implemented real-time console logging for parameter submissions:
  - [UI] Submit successful for <ComponentName> upon success.
  - [UI] Submit failed for <ComponentName>: <ErrorDetails> upon failure.
- **Startup View Cleanup:** Commented out intrusive .show() invocations for the startup wizard and initial compound selector overlays across mainApp.py and Landing_page.py.

Engineering Impact

Ensures parameter changes save cleanly without throwing uncaught exceptions, and provides developers with immediate submission diagnostics.

3.8 PR #81: Compound Selector Restored, Selected Compounds Panel Hidden by Default, Configurable Mixer/Splitter

- **Repository Link:** PR #81 on GitHub
- **Status:** Merged
- **Timeline:** Created: 2026-06-04 | Closed: 2026-06-05

Context & Problem Statement

Suppressing the compound selector on startup created friction for users starting new simulations. Concurrently, the persistent left panel crowded small screens, and Mixer/Splitter components remained hardcoded to fixed stream counts without visual port updates on the canvas.

Detailed Technical Implementation

- **Startup Sequence Refinement:** Re-enabled the automatic launch of ComponentSelector on application startup while keeping the wizard overlay suppressed in Landing_page.py.
- **Dock Visibility Management:** Configured selectedElementsDock to initialize in a hidden state, unchecking its corresponding actionViewSelectedElements menu item by default.
- **Dynamic Node Generation for Mixer & Splitter:**
    - Added a configuration window accessible by double-clicking Mixer or Splitter blocks.
    - Integrated a QComboBox dropdown allowing users to define the number of connection streams dynamically.
    - Programmed real-time node rendering on the canvas so that graphical connection sockets update to match the selected stream count.
- **Diagnostic Logging:** Added logging to capture failures during input_params_list operations.

Engineering Impact

Empowers users to model multi-stream mixing and splitting operations dynamically on the canvas.

3.9 PR #82: Fix OMC Path Error

- **Repository Link:** PR #82 on GitHub
- **Status:** Merged
- **Timeline:** Created: 2026-06-09 | Closed: 2026-06-09

Context & Problem Statement

The path to the OpenModelica Compiler executable (OMC.exe) was hardcoded with a specific version number. Users with newer or alternate OpenModelica installations received fatal "OMC.exe not found" errors.

Detailed Technical Implementation

- **Dynamic Executable Resolution:** Removed hardcoded compiler paths and version-specific strings. Refactored process invocation routines to dynamically locate and bind to any active OpenModelica compiler binary installed in the system environment.

Engineering Impact

Ensures cross-version portability, allowing the GUI to execute simulations against any standard OpenModelica installation.

3.10 PR #83: Fixed Equation Simulation Mode and Modelica 4.x / OMC Compatibility Issues

- **Repository Link:** PR #83 on GitHub
- **Status:** Merged
- **Timeline:** Created: 2026-06-09 | Closed: 2026-06-10

Context & Problem Statement

Equation (EQ) simulation mode failed across modern OpenModelica releases due to obsolete Modelica 3.2.3 MSL bindings, legacy model data definitions, outdated connector names, and unhandled compiler diagnostic streams.

Detailed Technical Implementation

- **OpenModelica Backend Compatibility:**
  - Replaced hardcoded Modelica 3.2.3 annotations with flexible uses(Modelica) bindings.
  - Implemented automated cleanup routines to delete obsolete or temporary .mo files before generating new simulation scripts.
  - Corrected file loading paths for flowsheet packages and resolved missing parameter Integer Nc and property array declarations in InitialGuess.mo, RaoultsLaw.mo, and MaterialStream.mo.
- **Modelica 4.x Language Modernization:**
  - Converted all 431 chemical compound definitions and GeneralProperties.mo from model to record types to match their role as passive data structures.
  - Standardized connector interface syntax from .inlet, .outlet, and .energy to standard .In, .Out, and .En.
  - Implemented automated runtime management to write and append Flowsheet entries to package.order.
  - Updated SI unit imports to import SI = Modelica.Units.SI; and import Cv = Modelica.Units.Conversions;.
  - Enforced each keywords for non-scalar array modifications, replaced dimensionless marker "-" with "1", and standardized unit brackets from [unit] to (unit).
- **Simulation Pipeline Housekeeping:**
  - Updated result file naming conventions to Simulator.Flowsheet.FlowsheetSimulation_res.csv.
  - Resolved system overdetermination by bypassing redundant property equations on output streams.

- Updated result matrix validation in Container.py to accept 3-row outputs produced during 1-step simulations.
- Integrated getErrorString() calls inside execution scripts to capture detailed compiler errors.

Engineering Impact

Restores Equation simulation capabilities and aligns the backend with modern OpenModelica and Modelica 4.x standards.

3.11 PR #84: Basicfix: Modelica 4.x Semantic Formatting

- **Repository Link:** PR #84 on GitHub
- **Status:** Merged
- **Timeline:** Created: 2026-06-10 | Closed: 2026-06-11

Context & Problem Statement

Modern OpenModelica compilers enforce strict type and unit checking, flagging non-standard dimensionless annotations and bracketed unit expressions as compilation warnings and errors.

Detailed Technical Implementation

- **Dimensionless Standard Compliance:** Replaced all occurrences of the deprecated "-" dimensionless unit marker with the standard "1" across all parameter definitions and mole fraction attributes.
- **Unit Syntax Modernization:** Converted illegal square brackets in unit expressions (such as [kmol.K]) to standard parentheses (kmol.K) across all affected .mo source files.

Engineering Impact

Eliminates unit-checking syntax warnings during flowsheet compilation.

3.12 PR #85: Fixed: OMC Parse Error (Expected RBRACE, got 'formate')

- **Repository Link:** PR #85 on GitHub
- **Status:** Merged
- **Timeline:** Created: 2026-06-10 | Closed: 2026-06-11

Context & Problem Statement

Simulating compounds with hyphens, spaces, or chemical prefixes (e.g., 'Carbon-tetrachloride', 'Hydrogen cyanide') triggered compiler syntax crashes (Expected RBRACE) because OpenModelica identifiers cannot contain hyphens or spaces.

Detailed Technical Implementation

- **Identifier Sanitization (Flowsheet.py):** Integrated the previously unused

_normalize_compound_name utility into flowsheet generation, programmatically stripping spaces, hyphens, and illegal characters to generate compliant Modelica identifiers (e.g., 'Carbontetrachloride').

- **String Parser Hardening (UnitOperations.py):** Replaced fragile strip('[').strip(']') string manipulation logic with _normalize_compound_name, preventing string corruption during multi-compound list processing.

Engineering Impact

Allows multicomponent systems containing complex or hyphenated chemical compound names to compile without syntax errors.

3.13 PR #86: Fix: GUI Results Tab Not Populating After Successful Simulation

- **Repository Link:** PR #86 on GitHub
- **Status:** Merged
- **Timeline:** Created: 2026-06-17 | Closed: 2026-06-17

Context & Problem Statement

Running an Equation-Oriented (EQN) simulation produced a "Simulation Successful" dialog, yet the GUI Results tab remained blank due to an asynchronous notification disconnect and uncalled data extraction routines.

Detailed Technical Implementation

- **Thread-Safe Notification Signals (Container.py):** Defined results_ready = pyqtSignal() inside SimulationSignals to emit an event once data writing is confirmed.
- **Automated Result Rendering:** Implemented _populate_results() in Container.py to trigger DockWidget.show_result() when the simulation completes.
- **Data Access Realignment:** Replaced deprecated Prop attribute lookups with current MaterialStream dictionary access via unitop.variables[key]['value'], and connected previously uncalled self.ext_data() routines.
- **Simulation Verification:** Added file existence checks (os.path.exists(csvpath)) to verify output data before updating the GUI.

Engineering Impact

Eliminates blank result screens by ensuring that simulation outputs populate the Results view immediately upon completion.

3.14 PR #88: Fix: Post-Simulation GUI Crashes & Results Parsing

- **Repository Link:** PR #88 on GitHub
- **Status:** Merged
- **Timeline:** Created: 2026-06-18 | Closed: 2026-06-18

Context & Problem Statement

Clicking on output stream components following simulation generated KeyError, ValueError, and C++ segmentation faults due to missing dictionary keys, stale keyword arguments, and unvalidated string-to-float conversions.

Detailed Technical Implementation

- **Missing Key Registration (Streams.py):** Registered missing Fm_p[2] (Liquid Mass Flow) and Fm_p[3] (Vapour Mass Flow) keys inside the MaterialStream.variables dictionary.
- **Equation Initialization Guarding:** Corrected deprecated keywords xm_pc and Fm_pc to xm_pcarr and Fm_pcarr in get_start_values(), and replaced direct dictionary access for MW_p[2] and Cp_p[2] with safe .get() fallbacks.
- **Defensive Output Parsing (Flowsheet.py, Container.py):** Added length validation in ext_data() to skip parsing when result sets contain fewer than 2 rows. Moved update_tooltip_selectedVar() inside the successful execution block to avoid reading stale or corrupt data.
- **Safe Rendering Helpers:** Introduced _safe_float() in Streams.py and _safe_result() with .isdigit() validation in DockWidgetMaterialStream.py to sanitize numeric strings before updating table cells.

Engineering Impact

Prevents crashes during post-simulation inspection and ensures robust result rendering.

3.15 PR #91: Fix: Simulation Failing Due to Issues with Temp Bound and Semantics

- **Repository Link:** PR #91 on GitHub
- **Status:** Merged
- **Timeline:** Created: 2026-06-22 | Closed: 2026-06-22

Context & Problem Statement

The non-linear solver frequently crashed during model initialization when guessing negative temperatures in $\log(T)$ functions or encountering division-by-zero conditions when flows initialized to zero. Additionally, continuous relational operators violated symbolic differentiation rules, and valid 3-row outputs triggered false "Simulation Failed" warnings.

Detailed Technical Implementation

- **Domain Boundary Protection (MaterialStream.mo):** Introduced a protected bounded variable $T_{safe} = \max(T, 1)$ across logarithmic and exponential thermodynamic functions, preventing domain evaluation crashes.
- **Division-by-Zero Safeguards (MaterialStream.mo):** Wrapped phase fraction and dew point summation denominators in noEvent(max(Denominator, 1e-12)), guaranteeing non-zero divisors during initialization.
- **Strict Typing Compliance (Mixer.mo):** Removed erroneous each qualifiers from scalar variables (Tout, Pout) to satisfy Modelica type rules.
- **Symbolic Sanitization (InitialGuess.mo):** Extracted inline threshold evaluations into a discrete protected function computeFractions to avoid using discrete operators directly on continuous Real variables.
- **Result Row Acceptance (Container.py):** Adjusted the simulation output validation threshold to accept ≥ 2 rows, properly recognizing 3-row outputs produced during standard numberOfIntervals=1 runs.

Engineering Impact

Stabilizes solver convergence during initialization and eliminates false-alarm failure warnings in the GUI.

3.16 PR #92: Fix: String Parsing Issues for Simulation Results

- **Repository Link:** PR #92 on GitHub
- **Status:** Merged
- **Timeline:** Created: 2026-06-23 | Closed: 2026-06-24

Context & Problem Statement

A flawed conditional check in CSV header parsing led to incorrect variable mapping across UI tree widgets. Additionally, tree views initialized in a collapsed state, requiring users to manually expand nodes to inspect stream properties.

Detailed Technical Implementation

- **Tree View Usability:** Added expandAll() calls on mTreeWidget, lTreeWidget, and vTreeWidget during Amounts tab initialization so that all stream properties are visible immediately.
- **Boolean Logic Fix:** Corrected the CSV parsing condition from if i.find('['): (which evaluated -1 as True) to if '[' in i:, ensuring phase-indexed variables are mapped correctly.

Engineering Impact

Ensures reliable CSV result mapping and displays stream properties in an expanded, scannable format by default.

3.17 PR #94: Fix: NRTL Model

- **Repository Link:** PR #94 on GitHub
- **Status:** Merged
- **Timeline:** Created: 2026-06-25 | Closed: 2026-06-25

Context & Problem Statement

Simulations using the NRTL thermodynamic package failed to compile due to typographical errors in package imports and missing variable declarations. Stale result files from prior successful runs were being read by the GUI, masking compilation failures with false success indicators.

Detailed Technical Implementation

- **Import Path & Scoping Corrections:** Fixed import path typos across NRTL.mo, GraysonStreed.mo, HeatExchanger.mo, and PoyntingCF.mo, replacing Thermodynamic_Functions with ThermodynamicFunctions. Added missing parameter Integer Nc declarations in NRTL.mo.
- **Stale Result Deletion (Flowsheet.py):** Added explicit file deletion of Simulator.Flowsheet.FlowsheetSimulation_res.csv prior to starting each simulation, ensuring that old output files cannot mask compiler errors.

Engineering Impact

Enables reliable compilation of the NRTL thermodynamic package and ensures simulation reports reflect actual solver outcomes.

3.18 PR #96: Refactor: Centralize DockWidget Logic and Disable Submit Button After Sim Run

- **Repository Link: PR #96 on GitHub**
- **Status: Closed**
- **Timeline: Created: 2026-06-27 | Closed: 2026-06-27**

Context & Problem Statement

Initial draft pull request created to explore centralizing shared dock widget methods into a single base class and disabling parameter modifications after a simulation execution.

Technical Scope & Resolution

- **Superseded Implementation:** Served as the initial structural prototype for refactoring repeated methods across component dock widgets.
- **Consolidation:** Closed immediately in favor of a cleaned and rebased submission under PR #97, which delivered the full BaseDockWidget architecture and read-only field locking.

Engineering Impact

Established the foundational design for the centralized UI architecture finalized in PR #97.

3.19 PR #97: Fix: Editability of Fields After Sim Run; Refactor: Centralize DockWidget Logic

- **Repository Link:** PR #97 on GitHub
- **Status:** Merged
- **Timeline:** Created: 2026-06-27 | Closed: 2026-06-28

Context & Problem Statement

Component dock widgets contained extensive code duplication across 8 separate files for basic operations like displaying errors, clearing results, and closing widgets. Furthermore, users could modify input fields while viewing converged results, creating inconsistencies between displayed parameters and output tables.

Detailed Technical Implementation

- **Centralized Base Class (DockWidget.py):** Created a unified BaseDockWidget base class and refactored all 8 component-specific docks (Mixer, Flash, MaterialStream, etc.) to inherit from it.
- **Method Deduplication:** Removed redundant implementations of show_error, closeEvent, clear_results, and show_result from individual component files, centralizing them in BaseDockWidget.
- **Read-Only State Management:** Implemented centralized set_read_only logic in BaseDockWidget, locking input parameter fields, dropdown selectors, and the submit button once a simulation completes.

Engineering Impact

Significantly reduces codebase duplication, simplifies the addition of new unit operation docks, and prevents accidental parameter edits during result inspection.

3.20 PR #100: Fix: Loading .sim Files

- **Repository Link:** PR #100 on GitHub
- **Status:** Merged
- **Timeline:** Created: 2026-07-07 | Closed: 2026-07-08

Context & Problem Statement

Opening saved .sim flowsheet project files caused application crashes due to stale method calls, failed to redraw connection lines between streams and unit operations, triggered C++ segmentation faults during canvas positioning, and displayed false socket validation errors.

Detailed Technical Implementation

- **Crash Resolution (mainApp.py):** Removed a stale call to an obsolete compound_selection method that triggered fatal AttributeError crashes upon opening files.
- **Bidirectional Connection Scanning (Graphics.py):** Rewrote load_canvas() connection rebuild logic to scan both input_stms and output_stms. Streams do not populate their own output_stms, so scanning both lists ensures all stream-to-unit links are redrawn.
- **C++ Segfault Prevention:** Replaced unstable Qt mapToScene() calls with a safe _socket_scene_center() helper utilizing bounding box math and QApplication.processEvents().
- **Validation Sanitization (Container.py):** Updated pre-simulation validation checks to compute total_in and total_out connections, recognizing that streams function as feed-only or product-only blocks without throwing false connection warnings.
- **UI Polish (Landing_page.py):** Automatically suppressed the ComponentSelector popup when opening an existing project.

Engineering Impact

Restores complete .sim project loading, reconstructs canvas connection topologies reliably, and eliminates false-positive validation warnings.

3.21 PR #101: Redesign Amounts Section with Composition Basis Dropdown and Phase Sub-tabs

- **Repository Link:** PR #101 on GitHub
- **Status:** Merged
- **Timeline:** Created: 2026-07-09 | Closed: 2026-07-10

Context & Problem Statement

The MaterialStream Results panel used a deeply nested QToolBox layout that made inspecting

multi-component, multi-phase streams difficult, requiring users to navigate complex hierarchies to compare stream flows and fractions.

Detailed Technical Implementation

- **Tab Layout Restructuring:**
  - Replaced the legacy QToolBox with two primary top-level tabs: **Amounts** and **Phase Properties**.
  - Added a **Composition Basis** dropdown at the top of the Amounts tab to switch between **Mole Fraction**, **Mass Fraction**, **Mole Flow**, and **Mass Flow**.
  - Added dedicated **Mixture**, **Liquid**, and **Vapour** sub-tabs beneath the dropdown, rendering compound data filtered strictly for the active basis.
  - Preserved the Phase Properties layout for backward compatibility.
- **UI Codebase Cleanup:** Migrated DockWidgetMaterialStream.ui to the tabbed layout, updated table mapping routines in DockWidgetMaterialStream.py, and removed stale references to deleted tree widgets (mTreeWidget, lTreeWidget, vTreeWidget).

Engineering Impact

Improves data readability and navigation across multi-phase stream results.

3.22 PR #104: Feat: Expand Material Stream Phase Properties Results

- **Repository Link:** PR #104 on GitHub
- **Status:** Merged
- **Timeline:** Created: 2026-08-02 | Closed: 2026-08-03

Context & Problem Statement

Key phase equilibrium metrics—including Molar Specific Heat, Average Molecular Weight, and per-compound Vapor Pressure—were calculated by backend models but omitted from the GUI Results panel due to missing UI dictionary mappings.

Detailed Technical Implementation

- **Stream Variable Definitions (Streams.py):**
  - Added phase array entries Cp_p[1..3] (Molar Specific Heat) and MW_p[1..3] (Average Molecular Weight) across Mixture, Liquid, and Vapour phases to self.variables.
  - Updated init_variables() to dynamically generate compound-indexed Pvap_c[i] entries matching the number of selected compounds (N_c).
  - Updated param_setter() to reset Pvap_c[i] values to None upon parameter re-submission.

    - Implemented serialization handling in get_start_values() to format compound vapor pressures into Modelica array syntax ({val1, val2, ...}).
- **UI Mapping (DockWidgetMaterialStream.py):** Added Cp_p, MW_p, and Pvap_c to the lookup dictionary p inside results_category(), enabling the GUI to populate these metrics in output tables.

Properties Registered & Mapped

Property MD	Modelica Variable MD	Scope / Indexing MD	Unit MD	Description MD
Molar Specific Heat	Cp_p[1..3]	3-Phase Array ([1]=Mixture, [2]=Liquid, [3]=Vapour)	kJ/(kmol.K)	Stream heat capacity
Average Molecular Weight	MW_p[1..3]	3-Phase Array ([1]=Mixture, [2]=Liquid, [3]=Vapour)	kg/kmol	Phase average molecular weight
Vapor Pressure	Pvap_c[1..Nc]	Compound Array (1 per selected component)	Pa	Pure-component saturated vapor pressure

Engineering Impact

Expands the thermodynamic output capabilities of the simulator, making heat capacity and vapor pressure data directly accessible in the interface.

3.23 PR #105: Fixed UI and UX of Phase Properties

- **Repository Link:** PR #105 on GitHub
- **Status:** Merged
- **Timeline:** Created: 2026-08-07 | Closed: 2026-08-07

Context & Problem Statement

The Phase Properties tab still used an accordion QToolBox layout that clashed with the tabbed design of the Amounts section. Furthermore, a data routing bug misallocated compound-indexed

vapor pressures (Pvap_c): because Pvap_c is indexed per compound ($[1..N_c]$), the standard 3-phase routing loop incorrectly placed Compound 2's vapor pressure into the Liquid table and Compound 3's into the Vapour table.

Detailed Technical Implementation

- **Modernized Tab Layout (DockWidgetMaterialStream.ui):** Replaced the QToolBox with a horizontal QTabWidget containing **Mixture**, **Liquid**, and **Vapour** sub-tabs, establishing visual consistency with the Amounts tab.
- **Dedicated Vapor Pressure Routing (DockWidgetMaterialStream.py):** Removed Pvap_c from the default 3-phase loop and implemented a dedicated extraction block in results_category() that appends all compound vapor pressure values exclusively to the **Mixture** table.
- **Nomenclature Cleanup (Streams.py):** Stripped internal (chemsep) database tags from labels, formatting them cleanly as Vapor Pressure(Argon) instead of Argon(chemsep) Vapor Pressure.

Engineering Impact

Eliminates property misallocation bugs across phase tables and aligns the visual design across stream result views.

3.24 PR #106: Enhancement: Implement Categorized Dropdown Filter for Phase Properties UI

- **Repository Link:** PR #106 on GitHub
- **Status:** Merged
- **Timeline:** Created: 2026-08-18 | Closed: 2026-08-18

Context & Problem Statement

The Phase Properties tab displayed all thermodynamic properties in a single long table, making it difficult to locate specific variables. The interface lacked the category filtering available in the Amounts tab, and the internal dictionary contained arbitrary property entries that did not match official specifications.

Detailed Technical Implementation

- **Categorized Dropdown UI (DockWidgetMaterialStream.ui):** Integrated a QComboBox at the top of the Phase Properties tab, mirroring the layout and user experience of the Composition Basis dropdown in the Amounts tab.

- **Dynamic Property Filtering (DockWidgetMaterialStream.py):**
    - Configured the dropdown with distinct property categories: **Vapor Pressure**, **Thermodynamic Properties** (Enthalpy, Entropy, Molar Specific Heat), **Molecular Weight**, **Mole Flow**, and **Mass Flow**.
    - Implemented the _on_phase_props_basis_changed signal handler to dynamically show and hide corresponding table rows across the **Mixture**, **Liquid**, and **Vapour** tables based on the selected category.
- **Strict Specification Mapping:** Cleaned the internal property dictionary p by removing arbitrary entries (Pressure, Temperature, and Vapour Phase Mole Fraction) so that tables strictly display specified metrics.

Engineering Impact

Completes the UI standardization of the MaterialStream Results panel, delivering a consistent and categorized inspection workflow across all stream properties.

CHAPTER 5: Conclusion

4.1 Tasks Accomplished

During the FOSSEE Summer Internship, contributions were made to the development, stability, and modernization of the **Chemical Simulator GUI**—an open-source chemical process simulation desktop platform powered by OpenModelica. Across 24 pull requests, the work addressed critical bugs, improved UI ergonomics, standardized backend Modelica packages, and stabilized solver routines.

Key achievements include:

- **Modelica 4.x Compliance & Modernization:** Modernized the backend thermodynamic library by converting 431 pure-component chemical files and GeneralProperties.mo from model classes to standard record data containers. Standardized connector interfaces (.In, .Out, .En), resolved SI unit import paths, updated dimensionless attributes to "1", and standardized unit expression syntax across all package definitions.
- **Solver Initialization & Domain Protection:** Hardened nonlinear equation solving in MaterialStream.mo against runtime domain crashes by introducing bounded temperature definitions ($T_{safe} = \max(T, 1)$) and preventing zero-division errors during initialization using noEvent(max(Denominator, 1e-12)). Sanitized discrete operators on continuous variables via protected calculation routines.
- **Flowsheet File (.sim) Loading Stability:** Resolved application crashes and C++ level segmentation faults during project loading. Rewrote canvas topology reconstruction in Graphics.py to scan both input and output streams, eliminated unhandled AttributeError exceptions, and corrected validation routines in Container.py to support unidirectional streams without emitting false connection warnings.
- **UI Architecture Refactoring:** Centralized shared UI logic from 8 distinct component files into a single BaseDockWidget base class. Standardized read-only field toggling across parameter inputs post-simulation, preventing UI state desynchronization while converged results are displayed.
- **Results Panel Redesign & Dynamic Filtering:** Re-engineered the MaterialStream Results panel with top-level tab splits for **Amounts** and **Phase Properties**. Added dynamic dropdown filters and dedicated sub-tabs (Mixture, Liquid, Vapour) for inspecting mole/mass fractions, flows, thermodynamic properties, and pure-component vapor pressures.

- **Thermodynamic Output Expansion:** Expanded simulation output extraction to parse and render Molar Specific Heat (C_{p_p}), Average Molecular Weight (MW_p), and per-compound Vapor Pressure (P_{vap_c}) with defensive validation helpers.

4.2 Skills Developed

4.2.1 Technical Skills

- **GUI Architecture & Qt Framework:** Advanced proficiency in Python and PyQt5, including QGraphicsScene canvas manipulation, QDockWidget lifecycle management, custom signal-slot event-driven communication, and .ui layout design.
- **Equation-Oriented Simulation & Modelica:** Deep understanding of Modelica 4.x language specifications, the OpenModelica Compiler (OMC) pipeline, script automation (.mos), connector semantics, and symbolic solver constraints for nonlinear algebraic systems.
- **Defensive Programming & Diagnostics:** Developing robust parsing routines for simulation CSV matrices, integrating safe type conversion helpers, isolating segmentation faults in C++ Qt bindings, and embedding runtime compiler diagnostics (getErrorString()).
- **State Management & Data Flow:** Structuring consistent data exchange between the graphical canvas, parameter dock widgets, backend file generation routines, and simulation execution threads.

4.2.2 Professional Skills

- **Open-Source Collaboration:** Managing feature development, atomic commits, branch rebasing, and pull requests within a structured GitHub open-source repository.
- **Code Review & Refactoring:** Identifying systemic architectural redundancy and consolidating duplicated logic into clean, reusable object-oriented design patterns.
- **Root-Cause Debugging:** Systematically tracing crashes across the Python-OMC-C++ boundary, creating reproducible test cases, and deploying regression-free patches.
- **Technical Documentation:** Authoring precise pull request descriptions, system architecture notes, and user-facing interface documentation.

4.3 Future Scope

The Chemical Simulator GUI provides an open-source foundation for process modeling. Promising avenues for future development include:

- **Advanced Unit Operations:** Extending the component library to include rigorous multi-stage distillation columns, absorption columns, plug-flow reactors (PFR), and continuous stirred-tank reactors (CSTR).
- **Dynamic Simulation Capabilities:** Expanding beyond steady-state calculations to support

transient, time-dependent dynamic flowsheet simulations and control loop modeling.

- **Thermodynamic Database Expansion:** Integrating additional equation-of-state models (such as Peng-Robinson, SRK, and UNIQUAC) and developing a graphical custom property package manager for user-defined pseudo-components.
- **Automated Flowsheet Diagnostics:** Implementing real-time degree-of-freedom analysis and topological consistency checks on the canvas prior to invoking the solver.
- **Cloud & Cross-Platform Packaging:** Establishing automated CI/CD build pipelines for cross-platform installers (Windows, macOS, Linux) and exploring containerized headless simulation execution.

Bibliography

[1] FOSSEE Project, *Free/Libre and Open Source Software for Education*, Indian Institute of Technology Bombay, Available: https://fossee.in/, Accessed: 2026.

[2] FOSSEE Chemical Simulator Team, *Chemical Simulator GUI Repository*, GitHub, Available: https://github.com/FOSSEE/Chemical-Simulator-GUI, 2026.

[3] Open Source Modelica Consortium (OSMC), *OpenModelica System Documentation and User's Guide*, Available: https://openmodelica.org/doc/OpenModelicaUsersGuide/latest/, Accessed: 2026.

[4] Modelica Association, *Modelica® – A Unified Object-Oriented Language for Systems Modeling: Language Specification Version 4.0*, Modelica Association Project, 2020.

[5] Riverbank Computing, *PyQt5 Reference Guide and Class Hierarchy Documentation*, Available: https://www.riverbankcomputing.com/static/Docs/PyQt5/, Accessed: 2026.

[6] Python Software Foundation, *Python 3 Language Reference and Subprocess Management API*, Available: https://docs.python.org/3/, Accessed: 2026.

[7] National Mission on Education through Information and Communication Technology (NMEICT), *Ministry of Education, Government of India*, Available: https://www.education.gov.in/, Accessed: 2026.