eSim Summer Fellowship 2026

On

Developing eSim on Cloud

Submitted by

Samay Rajput

B.Tech CSE (Gaming Tech), 3rd Year

VIT Bhopal University

Under the guidance of

Prof. Prabhu Ramachandran

Principal Investigator

Department of Aerospace Engineering

Indian Institute of Technology Bombay

July 28, 2026

Acknowledgment

I express my sincere gratitude to Prof. Prabhu Ramachandran for providing me with the opportunity to be part of the FOSSEE internship program and for his continued efforts in promoting open source engineering tool development. His leadership and vision have been instrumental in fostering meaningful student participation in the open-source ecosystem.

I also acknowledge Prof. Kannan M. Moudgalya for his foundational role in establishing and strengthening the FOSSEE initiative. His contributions to open source education and the development of the FOSSEE fellowship framework have been pivotal in creating the academic and organisational platform through which this internship was undertaken.

My sincere appreciation extends to my mentor, Sumanto Kar, for his continual support, technical guidance, and encouragement throughout the duration of this project. His insights and feedback played a key role in refining ideas, overcoming challenges, and ensuring timely completion of the tasks assigned to me.

I would also like to thank my internal mentors, Mr. Varad Patil and Ms. Shanthi Priya K for their valuable guidance, coordination, and technical inputs during the internship. Their mentorship contributed significantly to the clarity, progress, and successful execution of the work.

This internship has been an enriching learning experience, allowing me to work closely with open source EDA tools, contribute to the eSim-Cloud web platform, and gain exposure to real world frontend, simulation pipeline, and DevOps engineering workflows. The knowledge acquired during this period will undoubtedly support my future academic and professional pursuits.

I would also like to thank the entire FOSSEE team for their coordination, assistance, and timely interactions at various stages of this work. Their collective efforts ensured smooth workflow, resource accessibility, and effective project execution.

Contents

Chapter 1

Introduction

FOSSEE (Free/Libre and Open Source Software for Education) is an organization based at IIT Bombay, functioning as a remarkable initiative aimed at promoting the use of open source software in education and research. It was established with the mission to reduce dependency on proprietary software and to encourage the adoption of open source alternatives.

1.1 eSim-Cloud

eSim-Cloud is the web based evolution of the desktop eSim software developed by FOSSEE. Operating entirely within a standard web browser, it removes local hardware and operating system dependencies, making circuit design and SPICE based simulation accessible to students, educators, and researchers globally. The client side is built on ReactJS coupled with mxGraph, an industry standard diagramming library used for schematic capture and topological state management, while the server side is orchestrated using Django, Celery, and Redis.

1.2 Ngspice

Ngspice is the open source SPICE simulation engine used by eSim-Cloud to perform DC, transient, and AC (small signal frequency) analysis on user designed circuits. It computes complex nodal voltage vectors that the frontend is responsible for requesting, parsing, and rendering correctly.

1.3 Docker and Containerized Deployment

The entire eSim-Cloud infrastructure is containerized using Docker, with Alpine Linux base images used for the backend Django and Celery containers. This guarantees envi-

ronment parity between local development and production deployments, and allows for dynamic scalability.

Chapter 2

Features of eSim-Cloud

eSim-Cloud aims to provide a fully browser accessible EDA solution capable of schematic creation and analog circuit simulation without local software installation.

- **Schematic Creation:** An mxGraph based graphical canvas for drawing circuit schematics directly in the browser.
- **Circuit Simulation:** Native integration with the Ngspice engine, supporting DC Solver, DC Sweep, Transient Analysis, and AC Analysis modes.
- **Undo/Redo History:** A schematic editing history stack that allows users to reverse and reapply topological changes to the circuit.
- **Theming Support:** A Material UI driven Light/Dark mode toggle that governs the visual appearance of the entire application.
- **Cloud Native Deployment:** Dockerized Django, Celery, and Redis services enabling consistent, scalable deployment.

Chapter 3

Problem Statement

The objective of this fellowship phase was to systematically identify, isolate, and resolve underlying issues affecting the visualization accuracy, editor reliability, UI consistency, and build stability of the eSim-Cloud platform, across four distinct areas of the application: AC analysis data visualization, schematic editor history management, UI theming, and backend dependency resolution.

3.1 Approach

The work was carried out following a structured, task wise roadmap:

1. **Issue Identification:** Reviewing existing bug reports and manually reproducing each issue (incorrect Bode plots, corrupted undo/redo, theme-unresponsive Help screen, failing Docker builds).

2. **Root Cause Analysis:** Tracing each issue to its source in the codebase: frontend payload construction, mxGraph history heuristics, MUI styling definitions, and backend dependency versions respectively.

3. **Implementation:** Applying targeted code changes to `SimulationProperties.js`, `Graph.js`, `ToolbarTools.js`, `ToolbarExtension.js`, and `requirements.txt`.

4. **Verification:** Testing each fix against representative circuits and build scenarios to confirm correctness before submission.

Chapter 4

Tasks Undertaken

4.1 AC Analysis and Bode Plot Visualization

4.1.1 Description

The AC Analysis panel in eSim-Cloud previously requested the raw nodal voltage from Ngspice and plotted it on a linear axis, producing a non intuitive plot that failed to represent wide band frequency response. The task involved reworking the request payload in `SimulationProperties.js` and the charting logic in `Graph.js` to produce a mathematically correct, dual axis logarithmic Bode plot.

4.1.2 Technical Analysis

For a linear time invariant network with transfer function $H(j\omega)$, the Bode representation separates the response into:

$$\text{Magnitude (dB)} = 20\log_{10}|H(j\omega)|$$
$$\text{Phase} = \angle H(j\omega)$$

Ngspice natively computes these as `vdb(node)` and `vp(node)` respectively, but the frontend was requesting only the plain node voltage, discarding this information. In addition, frequency must be plotted on a logarithmic X axis to avoid compressing multiple decades of sweep data into a few pixels.

4.1.3 Challenges Encountered

- **Linear frequency compression:** Logarithmic sweep data rendered on a linear pixel scale squeezed low frequency behaviour into the extreme left edge of the chart.
- **Dataset disassociation:** Chart.js was using categorical label index positioning,

which misaligned data points once a logarithmic scale was introduced.

- **Floating point truncation:** The phase dataset was being truncated to a single decimal place by a generic precision formatter, causing jagged "staircase" artifacts in place of a smooth curve.

4.1.4 Solution and Implementation

The `startSimulate` function's `case 'Ac':` block was modified to explicitly request `vdb(node)` and `vp(node)` for the user selected node(s), and a node multi select dropdown was added to the AC Analysis panel to match the existing DC Sweep and Transient panels. In `Graph.js`, the X-axis scale type was set to logarithmic, dataset generation was changed to produce explicit `{x, y}` coordinate objects instead of relying on label indices, and the display precision parameter for the phase dataset was increased to three decimal places. Dual Y-axes were configured to separate magnitude (dB) and phase (degrees) without scale compression.

```
1     // AC Analysis controlline construction inside startSimulate()
2     case 'Ac':
3     if (acAnalysisControlLine.input !== '' && acAnalysisControlLine.
    pointsBydecade !== '' &&
4     acAnalysisControlLine.start !== '' && acAnalysisControlLine.stop
    !== '') {
5       typeSimulation = 'Ac'
6       controlLine = `.ac ${acAnalysisControlLine.input} ${
    acAnalysisControlLine.pointsBydecade} ${acAnalysisControlLine.
    start} ${acAnalysisControlLine.stop}`
7       dispatch(setResultTitle('AC Analysis Output'))
8       setSelectedValue(selectedValueAcAnal)
9     } else {
10      setNeedParameters(true)
11      return
12    }
13    break
14
15    // When building the .control print statement, AC Analysis
    requests
16    // both magnitude (dB) and phase for each selected node instead
    of
17    // the plain node voltage:
18    if (selectedValue.length > 0 && selectedValue !== null &&
    skipMultiNodeChk === 0) {
19      selectedValue.forEach((value, i) => {
20        if (value[i] !== undefined && value[i].key !== 0) {
21          atleastOne = 1
```

```
22            if (typeSimulation === 'Ac') {
23              cblockline = cblockline + ' vdb(' + String(value[i].key)
    + ') vp(' + String(value[i].key) + ')'
24            } else {
25              cblockline = cblockline + ' ' + String(value[i].key)
26            }
27          }
28        })
29      }
30
31      // Node multiselect dropdown added to the AC Analysis panel,
32      // matching the existing DC Sweep / Transient panels:
33      <Multiselect
34      style={{ width: '100%' }}
35      id="Nodes"
36      closeOnSelect="false"
37      placeholder="Select Node"
38      onSelect={handleAddSelectedValueAcAnal}
39      onRemove={handleRemSelectedValueAcAnal}
40      options={analysisNodeArray} displayValue="key"
41      avoidHighlightFirstOption="true"
42      />
43
```

Listing 4.1: Relevant changes in SimulationProperties.js

```
1       // Detect Bode-plot mode from the sweep type, and build explicit
2       // {x, y} coordinate objects instead of relying on label index
3       // positioning, so the logarithmic X-axis aligns correctly:
4       const isBode = labels[0] === 'frequency'
5       const xValues = x.map(e => Number((e / scales[xscale].value).
    toFixed(precision)))
6
7       const dataset = () => {
8         var arr = []
9         for (var i = 0; i < y.length; i++) {
10          if (labels[0] === labels[i + 1]) continue
11          const seriesLabel = labels[i + 1]
12          const lowerLabel = String(seriesLabel).toLowerCase()
13          const isPhase = lowerLabel.includes('vp(')
14          const entry = {
15            label: seriesLabel,
16            data: isBode
17            ? y[i].map((e, idx) => {
18              const val = e / scales[yscale].value
19              // Phase precision raised to 3 decimal places to remove
20              // the "staircase" artifact caused by 1 decimal
```

```
      truncation
21              const converted = isPhase ? val * (180 / Math.PI) : val
22              return {
23                x: xValues[idx],
24                y: Number(converted.toFixed(precision))
25              }
26            })
27            : y[i].map(e => (e / scales[yscale].value).toFixed(
      precision)),
28            fill: false
29          }
30          if (isBode) {
31            entry.yAxisID = lowerLabel.includes('vp(') ? 'y-phase' : 'y
      -mag'
32          }
33          arr.push(entry)
34        }
35        return arr
36      }
37
38      // X-axis switched to logarithmic scale for Bode plots
39      scales: {
40        xAxes: [{
41          type: isBode ? 'logarithmic' : 'category',
42          display: true,
43          scaleLabel: { display: true, labelString: selectLabel() },
44          ticks: {
45            maxTicksLimit: scales[xscale].ticks,
46            callback: isBode ? (value) => {
47              const log = Math.log10(Number(value))
48              return Number.isInteger(log) ? value : null
49            } : undefined
50          }
51        }],
52        // Dual Y-axes separate magnitude (dB) and phase (degrees)
53        // without scale compression:
54        yAxes: isBode ? [
55        {
56          id: 'y-mag',
57          type: 'linear',
58          position: 'left',
59          scaleLabel: { display: true, labelString: 'Magnitude ( dB )'
      }
60        },
61        {
62          id: 'y-phase',
63          type: 'linear',
```

```
64          position: 'right',
65          scaleLabel: { display: true, labelString: 'Phase ( degrees )'
        },
66          gridLines: { display: false }
67        }
68        ] : [
69        { id: 'y-mag', display: true, scaleLabel: { display: false,
      labelString: 'Voltage ( V )' } }
70        ]
71      }
72
```

Listing 4.2: Relevant changes in Graph.js

4.1.5 Testing and Verification

A standard first order RC low pass filter ($R = 1\,\text{k}\Omega$, $C = 10\,\mu\text{F}$, corner frequency ≈ 15.9 Hz) was used as the verification circuit, shown below with all connections and ground references in place.

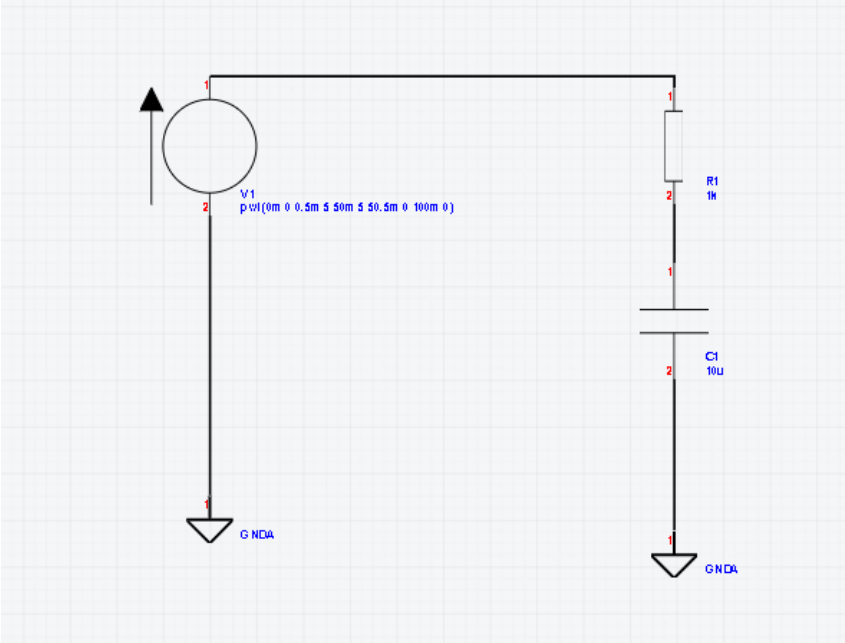

Figure 4.1: RC low pass filter test circuit used for AC Analysis verification, with grounds connected

The resulting Bode plot confirmed a flat magnitude response below the corner frequency, a smooth -20 dB/decade roll off thereafter, and a phase curve asymptoting toward $-\pi/2$ radians as frequency increased, with no visible staircase artifacts.

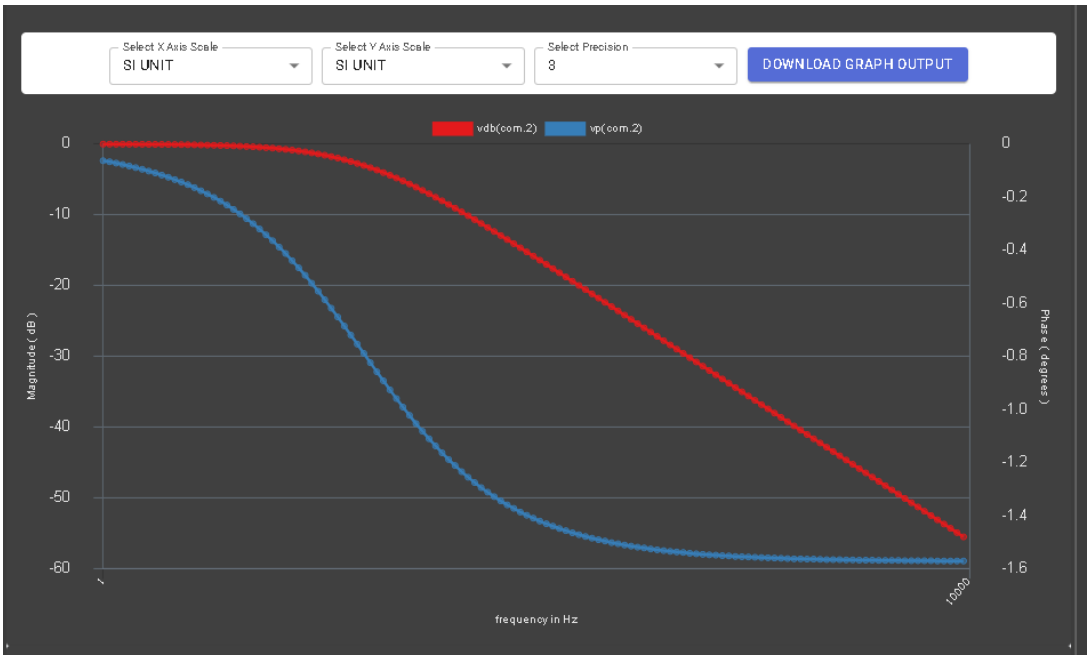

Figure 4.2: Dual axis logarithmic Bode plot showing magnitude (red, dB) and phase (blue, degrees) after the fix

4.2 Schematic Editor History Stack (Undo/Redo)

4.2.1 Description

The mxGraph based schematic editor exhibited state corruption when undoing a bulk deletion (e.g. a full `ClearGrid()` operation). Instead of restoring the entire circuit in one step, the graph would partially restore, reappearing wires without components, or components without their ground/terminal connections.

4.2.2 Technical Analysis

mxGraph's `mxUndoManager` bundles individual atomic changes (`mxChildChange`, `mxGeometryChange`, etc.) into a single `mxUndoableEdit` per logical user action. The custom grouping heuristic in `ToolbarTools.js` used the length of the `changes` array to decide whether an edit belonged to a larger, multipart action. A large bulk deletion produces an edit with a very large `changes.length`, which the heuristic misclassified as the beginning of a multistep component add operation, causing it to keep walking backward into unrelated prior edits and group them into the same undo step.

4.2.3 Challenges Encountered

Debugging required attaching to the `mxUndoManager.history` array during a live bulk deletion event to trace exactly where the grouping heuristic diverged from the user's intent, since the visual symptoms (partially restored circuits) did not by themselves reveal which specific edits were being incorrectly merged.

4.2.4 Solution and Implementation

The `ClearGrid()` and `DeleteComp()` functions were wrapped in explicit `beginUpdate()` / `endUpdate()` transactional blocks, and a new helper function, `markLastEditAtomic()`, was introduced to tag the resulting `mxUndoableEdit` with a custom `isAtomicUserAction` flag. The `Undo()` and `Redo()` iteration loops were updated to immediately halt the grouping heuristic upon encountering an edit carrying this flag, ensuring bulk actions are treated as a single, indivisible step.

```
1     // Atomic edit tagging: bulk operations (ClearGrid, DeleteComp)
      are
2     // wrapped in a single beginUpdate/endUpdate pair, guaranteeing
      exactly
3     // ONE mxUndoableEdit is produced. We tag that edit so Undo()/
      Redo()
```

```
    // can treat it as a single indivisible step, bypassing the legacy
    // grouping heuristic.
    function markLastEditAtomic() {
      const lastEdit = undoManager.history[undoManager.history.length - 1]
      if (lastEdit) {
        lastEdit.isAtomicUserAction = true
      }
    }

    // DELETE COMPONENT
    export function DeleteComp() {
      graph.getModel().beginUpdate()
      try {
        graph.removeCells()
      } finally {
        graph.getModel().endUpdate()
      }
      markLastEditAtomic()
    }

    // CLEAR WHOLE GRID
    export function ClearGrid() {
      graph.getModel().beginUpdate()
      try {
        graph.removeCells(graph.getChildVertices(graph.getDefaultParent()))
      } finally {
        graph.getModel().endUpdate()
      }
      markLastEditAtomic()
    }

    // UNDO (relevant branch added)
    export function Undo() {
      if (undoManager.indexOfNextAdd === 0) {
        return
      } else if (undoManager.history[undoManager.indexOfNextAdd - 1].isAtomicUserAction) {
        // Known-atomic edit: undo in a single step, skip the heuristic entirely
        undoManager.undo()
      } else if (checkWireChange(undoManager.history[undoManager.indexOfNextAdd - 1].changes)) {
        undoManager.undo()
      } else if (undoManager.history[undoManager.indexOfNextAdd - 1].
```

```
    changes.length > 1) {
45          let undos = 1
46          for (let i = undoManager.indexOfNextAdd - 1; i >= 0; i--,
    undos++) {
47            if (undoManager.history[i].changes.length === 1
48            || checkWireChange(undoManager.history[i].changes)
49            || undoManager.history[i].isAtomicUserAction
50            ) { break }
51          }
52          while (undos !== 0) { undoManager.undo(); undos-- }
53        }
54        // ...rest unchanged
55      }
56
57      // REDO mirrors the same isAtomicUserAction check
58      export function Redo() {
59        if (undoManager.indexOfNextAdd === undoManager.history.length)
    {
60          return
61        } else if (undoManager.history[undoManager.indexOfNextAdd].
    isAtomicUserAction) {
62          undoManager.redo()
63        }
64        // ...rest unchanged, with isAtomicUserAction added as a break
    condition
65        // in each grouping loop
66      }
67
```

Listing 4.3: Relevant changes in ToolbarTools.js

4.2.5 Testing and Verification

Before the fix, undoing a full grid clear restored the circuit incorrectly: wires and components reappeared without their proper connections, as shown below.

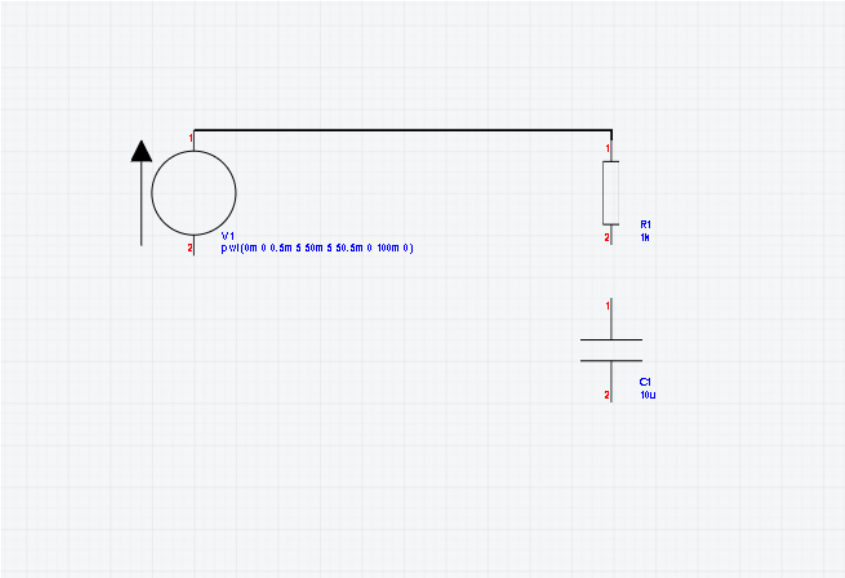

Figure 4.3: Partial, incorrect restoration after Undo: components restored without ground connections

After the fix, a single Undo correctly and completely restored the entire circuit, including all terminal and ground connections, in exactly one step.

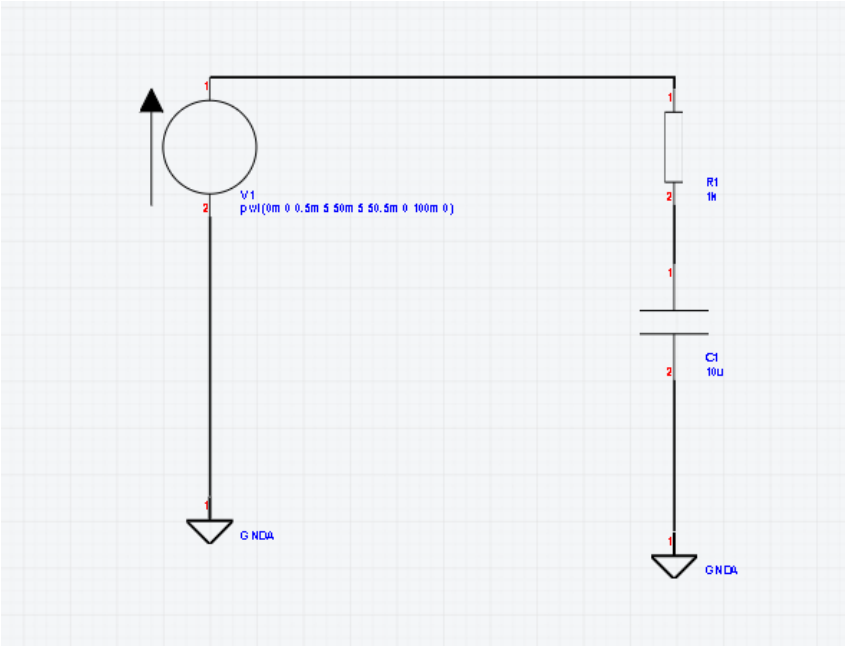

Figure 4.4: Complete, correct restoration after Undo: all components and grounds restored

4.3 UI Theming: Help Screen Redesign

4.3.1 Description

The `HelpScreen` component (rendering the Keyboard Shortcuts and Units Table panels) was visually unresponsive to the application's global Light/Dark mode toggle, remaining in a fixed, hardcoded appearance regardless of the active theme.

4.3.2 Technical Analysis

Material UI propagates a global `Theme` object through React's Context API via a `ThemeProvider`. Components using the `useStyles` hook can react to `theme.palette.mode` automatically, but only if their style definitions reference theme tokens. The `HelpScreen` instead used static hexadecimal color strings, fully decoupling it from the application's theme context.

4.3.3 Challenges Encountered

The layout was also visually dated: shortcuts were rendered as plain text separated by dividers rather than as a clear, scannable list, and the panels did not make efficient use of available screen width on desktop viewports.

4.3.4 Solution and Implementation

All hardcoded hex colors in the `useStyles` definition were replaced with theme aware tokens (`background.paper`, `text.secondary`, `divider`, `action.hover`). The layout was restructured into a responsive two column grid, and keyboard shortcuts were converted into a data driven list rendered using MUI `Chip` components to visually resemble physical keycaps.

```
1     // useStyles hook: hardcoded hex colors replaced with theme aware
2     // tokens so the Help screen reacts to the active Light/Dark
      theme
3     const useStyles = makeStyles((theme) => ({
4       helpDialog: {
5         backgroundColor: theme.palette.background.default
6       },
7       helpToolbar: {
8         backgroundColor: theme.palette.type === 'dark' ? theme.
      palette.grey[900] : theme.palette.primary.dark,
9         color: theme.palette.getContrastText(theme.palette.type === '
      dark' ? theme.palette.grey[900] : theme.palette.primary.dark)
10      },
```

```
11      helpSection: {
12        padding: theme.spacing(3, 3.5),
13        borderRadius: theme.spacing(1.5),
14        backgroundColor: theme.palette.background.paper,
15        border: `1px solid ${theme.palette.divider}`,
16        height: '100%',
17        transition: theme.transitions.create(['box-shadow', '
    transform'], { duration: 200 }),
18        '&:hover': {
19          boxShadow: theme.shadows[6]
20        }
21      },
22      shortcutRow: {
23        display: 'flex',
24        justifyContent: 'space-between',
25        alignItems: 'center',
26        padding: theme.spacing(1.1, 1),
27        borderRadius: theme.spacing(0.75),
28        transition: theme.transitions.create('background-color', {
    duration: 150 }),
29        '&:hover': {
30          backgroundColor: theme.palette.action.hover
31        }
32      },
33      keyChip: {
34        fontFamily: 'monospace',
35        fontWeight: 700,
36        letterSpacing: '0.02em',
37        backgroundColor: theme.palette.type === 'dark' ? theme.
    palette.grey[800] : theme.palette.grey[200],
38        color: theme.palette.text.primary,
39        border: `1px solid ${theme.palette.divider}`
40      },
41      unitsTable: {
42        backgroundColor: theme.palette.background.paper,
43        color: theme.palette.text.primary
44      },
45      modeBlock: {
46        padding: theme.spacing(1.25, 1),
47        borderRadius: theme.spacing(0.75),
48        transition: theme.transitions.create('background-color', {
    duration: 150 }),
49        '&:hover': {
50          backgroundColor: theme.palette.action.hover
51        }
52      }
53    }))
```

```
54
55      // Keyboard shortcuts converted to a data driven array, rendered
        as
56      // MUI Chip components styled to resemble physical keycaps:
57      const shortcuts = [
58      { label: 'Undo', keys: ['Ctrl', 'Z'] },
59      { label: 'Redo', keys: ['Ctrl', 'Shift', 'Z'] },
60      { label: 'Zoom In', keys: ['Ctrl', '+'] },
61      { label: 'Zoom Out', keys: ['Ctrl', '-'] },
62      { label: 'Save Circuit', keys: ['Ctrl', 'S'] },
63      { label: 'Export as Image', keys: ['Ctrl', 'Shift', 'E'] },
64      { label: 'Clear All', keys: ['Shift', 'Del'] }
65      // ...remaining shortcuts
66      ]
67
68      {shortcuts.map((shortcut, index) => (
69        <React.Fragment key={shortcut.label}>
70        <div className={classes.shortcutRow}>
71        <Typography variant="body1" className={classes.shortcutLabel}>
72        {shortcut.label}
73        </Typography>
74        <div className={classes.keyGroup}>
75        {shortcut.keys.map((key, i) => (
76          <React.Fragment key={key}>
77          {i > 0 && <Typography component="span" className={classes.
        keyPlus}>+</Typography>}
78          <Chip label={key} size="small" className={classes.keyChip} />
79          </React.Fragment>
80          ))}
81        </div>
82        </div>
83        {index < shortcuts.length - 1 && <Divider />}
84        </React.Fragment>
85        ))}
86
87      // Layout restructured into a responsive two-column grid
88      // (Keyboard Shortcuts | Units Table), with Simulation Modes
89      // spanning the full width below:
90      <Grid container spacing={3} direction="row" justify="center"
        alignItems="stretch">
91      <Grid item xs={12} md={6}>
92      <div className={classes.helpSection}> {/* Keyboard Shortcuts */}
        </div>
93      </Grid>
94      <Grid item xs={12} md={6}>
95      <div className={classes.helpSection}> {/* Units Table */} </div>
96      </Grid>
```

```
97      <Grid item xs={12}>
98      <div className={classes.helpSection}> {/* Simulation Modes */} </
        div>
99      </Grid>
100     </Grid>
101
```

Listing 4.4: Relevant changes in ToolbarExtension.js (HelpScreen styling)

4.3.5 Testing and Verification

Before the fix, the Help panels retained a flat, generic grey appearance that did not visually integrate with the rest of the application in either mode:

Figure 4.5: Units Table and Simulation Modes panels prior to the fix: static, theme unaware styling

Figure 4.6: Keyboard Shortcuts panel prior to the fix: static, theme unaware styling

After the fix, the Help screen correctly inherits the active theme in both Dark and Light mode:

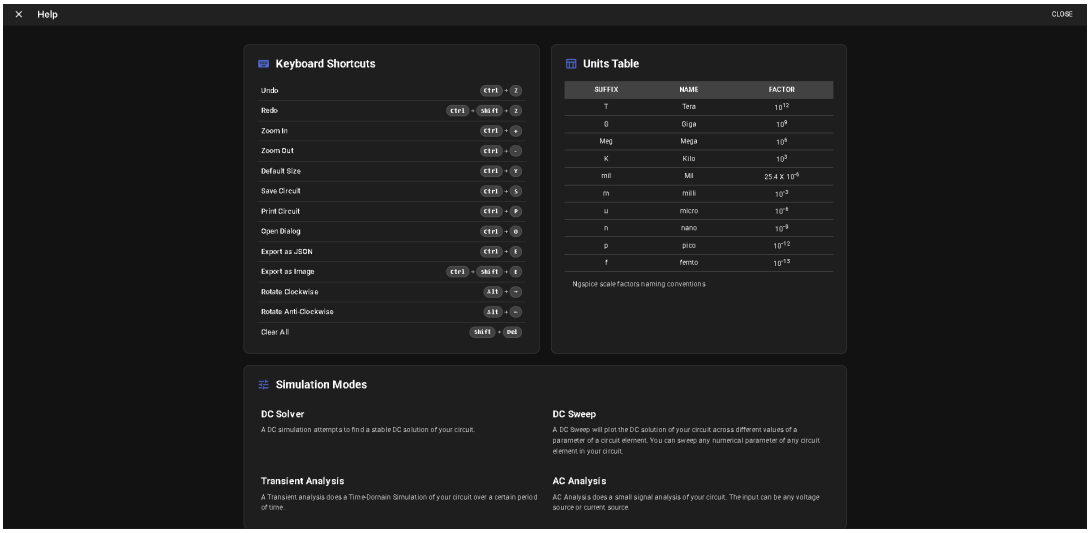

Figure 4.7: Redesigned Help screen in Dark mode

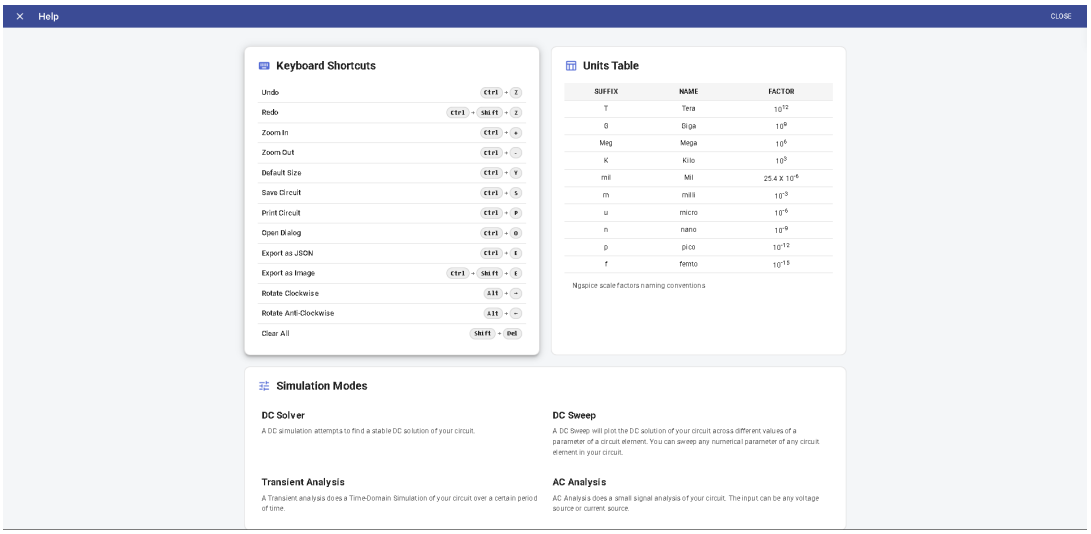

Figure 4.8: Redesigned Help screen in Light mode, confirming correct theme reactivity

4.4 Docker Build Pipeline: NumPy/Cython Dependency Resolution

4.4.1 Description

Fixed the Docker build failure caused by an incompatibility between the pinned NumPy version and newer Cython releases. The development environment setup was failing during the dependency installation step while building the Django and Celery Docker images.

4.4.2 Technical Analysis

The backend containers use Alpine Linux, which is built on the `musl` C library rather than `glibc`. Standard `manylinux` Python wheels assume `glibc` and are therefore incompatible with Alpine. When `pip` could not find a pre-built Alpine wheel for the pinned `numpy==1.18.4`, it fell back to a source build, which required Cython to transpile C extensions. Because `numpy==1.18.4` did not pin a compatible Cython version, dependency resolution was free to pull in a newer Cython 3.0+ release, which introduced breaking syntax changes incompatible with NumPy 1.18.4's C-API build scripts, causing the compilation step to fail.

4.4.3 Challenges Encountered

Several resolution strategies were evaluated: pinning Cython below 3.0, switching the base image to a Debian/glibc based one, or forcing prebuilt wheels only, each carrying trade offs in long term maintainability, image size, or risk of outright failure if no compatible wheel existed.

4.4.4 Solution and Implementation

The `numpy` version in `esim-cloud-backend/requirements.txt` was updated from `1.18.4` to `1.21.6`. NumPy 1.21.x carries the correct Cython version constraint, so dependency resolution no longer pulls in an incompatible Cython release, resolving the conflict without requiring any Dockerfile changes or additional `pip` flags.

```
1     # Before: numpy==1.18.4 does not pin a compatible Cython
2     # version, allowing dependency resolution to install an
3     # incompatible newer Cython release and break the build.
4     - numpy==1.18.4
5
6     # After: NumPy 1.21.x carries the required Cython version
7     # constraint, resolving the conflict with no Dockerfile
```

```
8       # changes or extra pip flags needed.
9       + numpy==1.21.6
10
```

Listing 4.5: Relevant change in requirements.txt (esim-cloud-backend)

4.4.5 Testing and Verification

- Rebuilt the Django Docker image successfully.
- Rebuilt the Celery Docker image successfully.
- Verified that the development containers build and start correctly using `docker-compose.dev.yml`.

Files Changed: `esim-cloud-backend/requirements.txt`

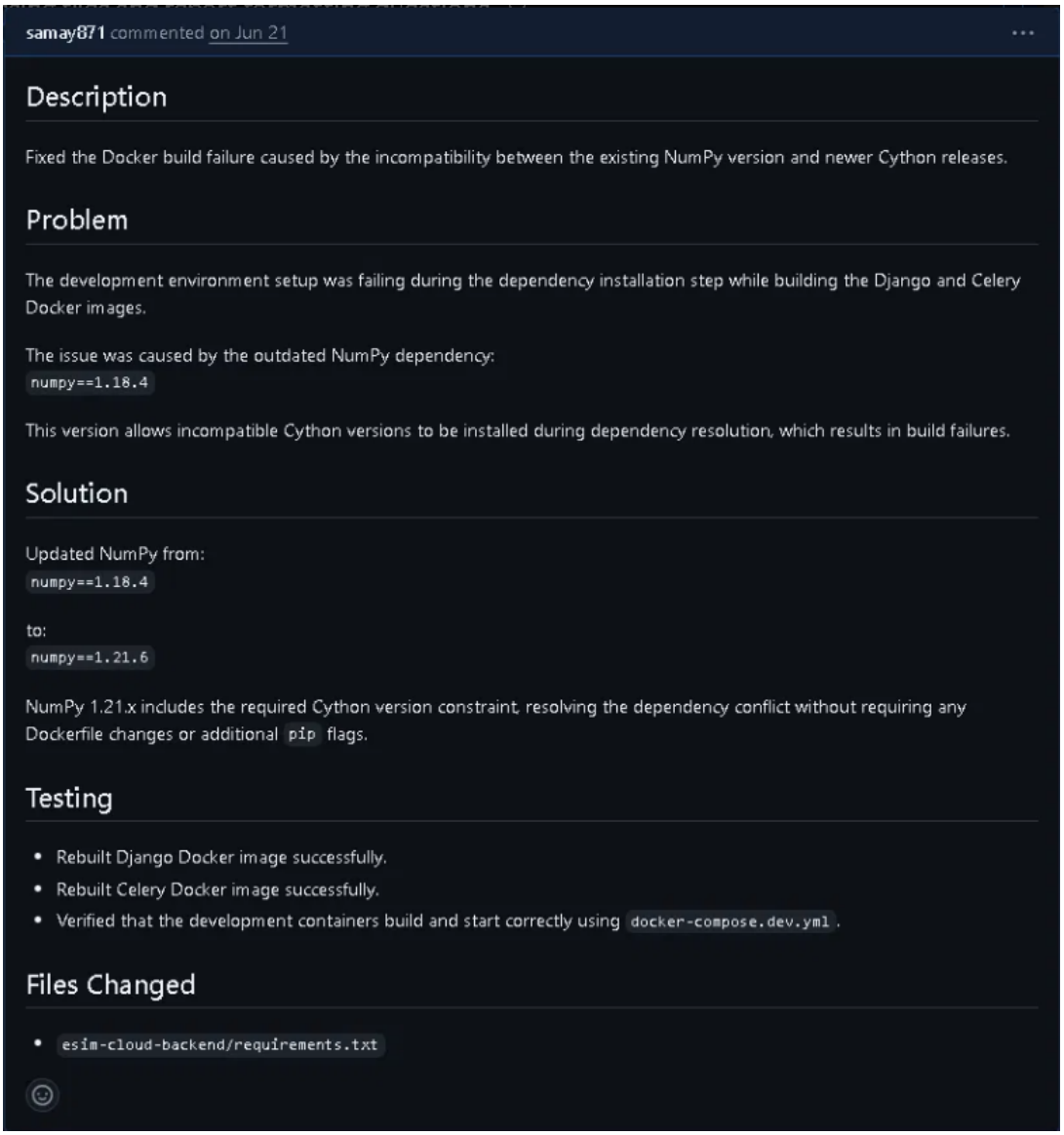

Figure 4.9: Pull request describing the NumPy version fix for the Docker build failure

Chapter 5

Skills and Knowledge Gained

Execution of these tasks required the synthesis of several advanced software engineering, systems administration, and electrical engineering concepts:

- **Advanced SPICE Simulation:** Gained deep insight into how Ngspice outputs complex frequency domain arrays, and how to parse these arrays for external charting libraries using the specific decibel and phase syntaxes `vdb()` and `vp()`.
- **JavaScript State Machines:** Acquired practical experience with the Command pattern and memento based history stacks through the manipulation of the mxGraph `mxUndoManager` and its underlying model arrays.
- **React Context and CSS-in-JS:** Learned to inject global state parameters into isolated component styles using Material UI's dynamic theming engine, while adhering to accessibility standards for contrast and readability.
- **Containerized DevOps and Build Toolchains:** Developed a low level understanding of Python packaging architecture (wheels versus source distributions), C extensions, Cython transpilation, and the fundamental differences between `glibc` and `musl` libc in Alpine Linux environments.

Chapter 6

Conclusion and Future Scope

Over the course of this fellowship phase, four distinct issues spanning the simulation visualization pipeline, the schematic editor's state management, the UI theming layer, and the backend build infrastructure of eSim-Cloud were identified, root caused, and resolved. The AC Analysis pipeline now produces mathematically accurate, publication ready Bode plots; the schematic editor's undo/redo stack reliably restores bulk operations in a single step; the Help screen correctly and consistently reflects the application's active theme; and the Docker build pipeline has been stabilized against the underlying Cython/NumPy incompatibility.

The following improvements are recommended as natural extensions of the work completed in this phase:

1. **WebGL accelerated waveform rendering.** The AC Analysis Bode plot fix could be extended by migrating the underlying charting engine to a WebGL accelerated renderer, allowing very large frequency sweep datasets (fine step sweeps across many decades) to be plotted without the frame rate degradation a Canvas based renderer like Chart.js exhibits at high point counts.

2. **Streaming simulation results over WebSockets.** Long transient or fine resolution AC simulations currently return their full result only once Ngspice has finished. Streaming partial results to the frontend as they become available over a WebSocket connection would let a user see a Bode plot or transient waveform take shape in real time rather than waiting on a single blocking response.

3. **Generalizing atomic edit tagging beyond `ClearGrid` and `DeleteComp`.** The `isAtomicUserAction` flag introduced to fix the undo/redo grouping heuristic currently covers only the two operations known to trigger the bug. Any other bulk operation added to the schematic editor in the future (for example, a bulk copy paste or a multi component alignment tool) should adopt the same

`beginUpdate()`/`endUpdate()` plus `markLastEditAtomic()` pattern from the outset, rather than rediscovering the same grouping bug independently.

4. **Extending theme awareness beyond the Help screen.** The approach used to retrofit `HelpScreen`, systematically replacing hardcoded hex values with the nearest semantically correct Material UI theme token, is directly reusable for any other legacy component in the application still found to be theme unaware, and would be a natural follow up audit task for a future fellow.

5. **A private Alpine wheel artifact registry.** On the DevOps side, maintaining a private artifact registry of prebuilt, Alpine compatible (`musl`) Python wheels for the backend's pinned dependencies would help immunize the Docker build pipeline against similar upstream regressions in the future, rather than relying on individual version bumps each time an upstream package temporarily lacks a compatible wheel.

6. **Automated regression coverage for all four fixes.** None of the four fixes in this phase currently has an automated regression test guarding against reintroduction (each was verified manually against a representative circuit or build scenario). Adding a lightweight automated check for each, for example a snapshot test asserting the AC Analysis payload requests `vdb()`/`vp()`, or a CI step that rebuilds the backend image on every pull request, would convert this phase's manual verification work into a lasting guardrail for the codebase.