



**MIDDLE EAST TECHNICAL UNIVERSITY**  
**ELECTRICAL-ELECTRONICS ENGINEERING DEPARTMENT**

**EE374**

**EE374 Project Report**

**Group 36**

**Mustafa Alp Ekici**  
**2521524**

**Emirhan Aydın**  
**2515559**

# 1 Introduction

## 1.1 Purpose and Importance of the Project

Electrical distribution networks are supported by power cables that carry energy from substations to residences, businesses, and public spaces. Selecting the proper cable type and size is critical to ensuring safe operation, reducing energy losses, and preventing costly failures or overheating. By comparing installation costs with energy losses over the cable's lifetime, the selection process ensures both regulatory compliance and long-term cost savings.

This project presents a software tool that automates cable selection based on user-defined load characteristics, environmental conditions, and cable configurations. By integrating engineering calculations into an easy-to-use interface, the tool helps electrical engineers and technicians make reliable, efficient, and economical decisions.

## 1.2 Scope and Objectives of the Project

The Power Cable Selection Interface encompasses the design, implementation, and validation of a Python-based application with a graphical user interface (GUI). The main features include:

- Developing a user-friendly GUI that allows users to enter various design parameters and configurations.
- Algorithms to calculate current rating, voltage drop, real (P) and reactive (Q) power losses, and voltage regulation.
- Temperature and trench number correction factors in accordance with international cable standards.
- A 10-year economic analysis combining cable cost and energy losses at a specified unit price.

The objectives of the project are:

1. Develop a Python application that accurately computes cable selection parameters from user inputs.
2. Design an intuitive GUI that guides users through data entry and clearly presents results.
3. Implement and verify calculation algorithms using manufacturer data and real-world scenarios.
4. Perform a long-term economic assessment to compare installation costs with energy loss expenses.

## 2 Project Concept

The Power Cable Selection Interface integrates a graphical user interface, a calculation engine, a data repository, and a reporting module into a single application. Users enter load parameters, environmental conditions, cable configuration, and system voltage through the GUI.

The calculation engine retrieves cable specifications and correction factors from the data repository, validates inputs, and computes cable ratings, voltage drop, real and reactive power losses, voltage regulation, and a ten-year cost estimate. Results are displayed immediately as tables and charts.

## 3 Program/Code Explanation

This chapter provides a structured review of the Python application. It starts with the design and functionality of the graphical user interface (GUI), followed by the process of importing and editing cable data. Then, the smart listing algorithm that filters suitable cable options according to user-defined parameters and technical constraints is explained in detail. Finally, the chapter covers the calculations responsible for evaluating line losses, voltage regulation, and performing a ten-year economic analysis.

### 3.1 GUI Design

This section presents the graphical user interface (GUI) layout developed to facilitate user interaction with the cable selection application. The main window is structured using a vertical layout with a fixed resolution of 1000×700 pixels, providing an organized and user-friendly interface.

User inputs are categorized and grouped into three vertical sections to enhance readability and workflow clarity:

- **Load Characteristics:** This section includes input fields for selecting the load type and specifying the active power (kW), reactive power (kVAR), and line-to-line voltage (V).
- **Environmental Conditions:** A drop-down menu is provided for users to select the ambient temperature, which affects current-carrying capacity via thermal correction factors.
- **Cable Configurations:** This section allows users to define the cable core type (single-core or three-core), placement method (flat or trefoil, applicable only for single-core), the number of parallel circuits, and the total cable length (in meters).

Below the input sections, two command buttons are included to initiate the application's core functions:

- *List Suitable Cables* — activates the smart listing algorithm to filter the cable database based on user-defined constraints.
- *Calculate* — performs technical and economic evaluations of the selected cable, including line losses, voltage regulation, and cost analysis.

### 3.2 Excel Loader Function

The Excel loader function imports the cable dataset into the application using the pandas library. To ensure compatibility with both development and compiled environments (via PyInstaller), the base path is determined dynamically using `sys._MEIPASS` or the script's directory. The full file path is then constructed, and the Excel sheet named "Sayfa1" is read, skipping the first row due to its blank contents. Column headers are manually assigned to standardize the dataset for later processing. In case of any loading errors, a try-except block displays a critical message and safely terminates the program.

```
try:
    base_path = getattr(sys, '_MEIPASS', os.path.dirname(__file__))
    file_path = os.path.join(base_path, 'EE374 Project Cable List.xlsx')

    self.cable_data = pd.read_excel(file_path, sheet_name='Sayfa1', skiprows=1) #
        Due to first blank row
    self.cable_data.columns = [
        'Cable ID', 'Cable Code', 'Voltage Level',
        'Current Capacity (A) - Flat', 'Current Capacity (A) - Trefoil',
        'Resistance (ohm/km)', 'Inductance (mH/km) - Flat',
        'Inductance (mH/km) - Trefoil', 'Capacitance (uF/km)', 'Price (TL/km)'
    ]

except Exception as e:
    QMessageBox.critical(self, 'Error', f'Failed to load cable data: {e}')
    sys.exit()
```

### 3.3 Smart Listing Algorithm

The Smart Listing algorithm begins by reading the user's inputs. However, in case of any errors, try-except block is used as shown below.

```
try:
    P = float(self.P_input.text()) # Active Power
    Q = float(self.Q_input.text()) # Reactive Power
    V = float(self.Voltage_input.text()) # Line-to-Line Voltage
    T_str = self.temperature_input.currentText()
    T = int(T_str.replace('C', ''))
    num_circuits = int(self.circuit_input.currentText()) # Number of Circuits
```

First, the apparent power  $S$  is calculated from the active power  $P$  and reactive power  $Q$ , and then the line current  $I_{load}$  is determined.

```

S = (P ** 2 + Q ** 2) ** 0.5 # Apparent power in kVA
I_load = (S * 1000) / (3 ** 0.5 * V) # Line current in Amps

```

Based on the voltage level entered by the user, the application filters the cable database (originally provided in an Excel file) to include only those cables whose insulation ratings are appropriate for the specified system voltage. Since the voltage ratings in the dataset are expressed in a "lower/upper" format (e.g., "3.6/6" kV), the application first parses this field into separate numeric lower and upper voltage values in volts:

```

#Parse and filter voltage levels
voltage_parts = cables['Voltage
Level'].str.extract(r'(?P<Lower>\d+(?:.\d+)?)/(?P<Upper>\d+(?:.\d+)?)')
cables['Lower Voltage (V)'] = voltage_parts['Lower'].astype(float) * 1000
cables['Upper Voltage (V)'] = voltage_parts['Upper'].astype(float) * 1000

```

The standard voltage levels considered are typically in the set 0.6/1 kV, 3.6/6 kV, 6/10 kV, 12/20 kV, 20.3/35 kV. To ensure safe operation, the program eliminates any cable whose upper voltage rating is below the system voltage. Among the remaining candidates, the subset with the closest upper voltage rating (i.e., the smallest rating that still satisfies the system requirement) is selected to provide a focused and relevant listing:

```

#Only accept cables with upper voltage rating >= system voltage, and apply safety margin
cables = cables[cables['Upper Voltage (V)'] >= V]
closest_voltage = cables['Upper Voltage (V)'].min()
cables = cables[cables['Upper Voltage (V)'] == closest_voltage]

```

Following voltage compatibility filtering, the list is further narrowed based on the selected core type and placement method. These parameters are used to determine which column of current-carrying capacity to consider, and to validate the user's configuration against physical constraints defined in the project specification. In particular, the number of allowable parallel circuits is limited by the trench correction table, which only supports up to six cables per trench. For three-core cables, up to six circuits are permitted. For single-core cables, a three-phase set requires three individual conductors per circuit. Hence, a maximum of two circuits ( $2 \times 3 = 6$  cables) is allowed.

The application enforces these restrictions and provides appropriate warning messages if the user exceeds the allowed limits. Additionally, the placement field is locked to 'Trefoil' for three-core cables (as placement has no impact in this configuration), while for single-core cables, the user may select either 'Flat' or 'Trefoil' placement manually:

```

if core_type_selected == 'Three-Core':
cables = cables[cables['Cable Code'].str.startswith('3x')]
capacity_col = 'Current Capacity (A) - Trefoil' # 3-core values come from Trefoil column
per Excel
if num_circuits > 6:
QMessageBox.warning(self, 'Limit Exceeded',
'Max 6 parallel circuits allowed for three-core cables.')
return

```

```

elif core_type_selected == 'Single-Core':
cables = cables[cables['Cable Code'].str.startswith('1x')]
if num_circuits > 2:
QMessageBox.warning(self, 'Limit Exceeded',
'Max 2 parallel circuits allowed for single-core cables.')
return
if placement_type == 'Flat':
capacity_col = 'Current Capacity (A) - Flat'
else:
capacity_col = 'Current Capacity (A) - Trefoil'

```

Next, temperature and trench correction factors are applied to adjust the required current capacity for environmental conditions. A temperature correction factor is looked up from a predefined dictionary using temp correction, and a trench correction factor. The ambient temperature correction is applied by dividing the calculated line current by the temperature factor from the lookup table. For trench grouping, the algorithm first checks the cable core type. For three-core cables, the trench correction factor is equal to the number of parallel circuits specified by the user. For single-core cables, the factor is calculated as three times the number of circuits and reflects the three-phase nature of the system; forming a three-phase circuit with single-core conductors requires three single core cables. The code that calculates the current capacity according to the temperature and trench number is as follows:

```

I_required = I_load / temp_factor

if core_type_selected == 'Three-Core':
    trench_factor = trench_correction.get(num_circuits, 1.00)
    cables['Effective Capacity'] = cables[capacity_col] * num_circuits * trench_factor *
        3

elif core_type_selected == 'Single-Core':
    trench_factor = trench_correction.get(3 * num_circuits, 1.00)
    cables['Effective Capacity'] = cables[capacity_col] * num_circuits * trench_factor

cables = cables[cables['Effective Capacity'] >= I_required]

```

After applying all relevant filters, including voltage level compatibility, core type, placement configuration, and depleted current capacity, if no cables remain in the list, the application proactively notifies the user. A message box is displayed indicating that no suitable cable could be found under the specified conditions:

```

if len(cables) == 0:
QMessageBox.information(self, 'No Cable Found', 'No cable satisfies the given
    conditions.')
return

```

Once the filtering process identifies a set of suitable cables, the application dynamically populates a table within the graphical user interface to present the results to the user. Each row in the table corresponds to a valid cable option, and each column displays the respective technical attributes derived from the dataset. The following code illustrates how the table is constructed and filled:

```

Show table
self.table.setRowCount(len(cables))
self.table.setColumnCount(len(cables.columns))
self.table.setHorizontalHeaderLabels(cables.columns)

for i in range(len(cables)):
    for j in range(len(cables.columns)):
        self.table.setItem(i, j, QTableWidgetItem(str(cables.iloc[i, j])))

self.table.resizeColumnsToContents()

```

### 3.4 Loss Calculation & Voltage Regulation Algorithm

Once the cable is selected from the filtered table, the application enables users to calculate key performance parameters including line losses, inductance and capacitance effects, and voltage regulation by utilizing medium- and short-line transmission models.

As in the smart listing algorithm, the loss calculation and voltage regulation algorithm starts by taking user input.

```

try:
    selected_row = self.table.currentRow()
    if selected_row == -1:
        QMessageBox.warning(self, 'Warning', 'Please select a cable first!')
        return

    # User Inputs
    cables = self.cable_data.copy()
    length = float(self.length_input.text())
    load_type = self.load_type.currentText()
    P_input = float(self.P_input.text())
    Q_input = float(self.Q_input.text())
    V = float(self.Voltage_input.text())
    T_str = self.temperature_input.currentText()
    T = int(T_str.replace('C', ''))
    num_circuits = int(self.circuit_input.currentText())
    core_type_selected = self.core_type.currentText()
    placement_type = self.placement_type.currentText()

```

After receiving the user inputs, the apparent power  $S$  is calculated from  $P$  and  $Q$ . Then the load current  $I_{load\_mag}$  is determined using  $S$  and  $V$ . The resistance, inductance, and capacitance values were then determined using the cable specifications dataset. Since the resistance values in the cable dataset are given in ohms per kilometer, the total resistance is calculated by multiplying this value by the cable length in kilometers. Since the calculated resistance value applies to a single-phase conductor, it is divided by the number of parallel circuits in that phase to determine the effective resistance per phase. The following code illustrates this calculation:

```

resistivity_per_km = float(cable['Resistance (ohm/km)'])
length_km = length / 1000

```

```
resistance = resistivity_per_km * length_km / num_circuits
```

The inductance calculation distinguishes between two cable placement types. The per-kilometer inductance value is retrieved directly from the cable specifications dataset: when the cable is arranged in a trefoil configuration, the trefoil inductance (mH/km) is used; when it is arranged flat, the flat inductance (mH/km) from the dataset is applied. Since three-phase systems with three-core cables require a trefoil arrangement, a flat inductance value is provided only for single-core cables in the dataset. The inductive reactance is calculated by taking the per-kilometre inductance value (in mH/km) from the cable specifications dataset and multiplying it by the cable length in kilometres and by  $100\pi$  to convert it into reactance at 50 Hz. The result is then divided by 1000 to convert millihenrys to henrys and divided by the number of parallel circuits to obtain the reactance per phase. The following code illustrates this calculation:

```
inductance = inductivity_per_km * length_km * 100 * math.pi / (1000 * num_circuits)
```

The capacitance ( $\mu\text{F}/\text{km}$ ) is retrieved from the cable specifications dataset in the same way as resistance and inductance. If a cable entry lacks a capacitance value, it is defaulted to zero. The capacitive susceptance is obtained by taking the capacitance ( $\mu\text{F}/\text{km}$ ) from the cable specifications dataset and multiplying it by the angular frequency  $100\pi$  rad/s, the cable length in kilometres, the factor  $10^{-6}$  to convert microfarads to farads, and the number of parallel circuits. This yields the total capacitive susceptance per phase. The following code illustrates this calculation:

```
if (pd.isna(capacitance_per_km_raw) or
    str(capacitance_per_km_raw).strip() == '' or
    str(capacitance_per_km_raw).lower() == 'nan'):
    capacitance_per_km = 0
    capacitance = 0
else:
    try:
        capacitance_per_km = float(capacitance_per_km_raw)
        capacitance = capacitance_per_km * length_km * 1e-6 * num_circuits
    except (ValueError, TypeError):
        capacitance_per_km = 0
        capacitance = 0

susceptance = 2 * math.pi * 50 * capacitance
```

With the line parameters determined, the focus now turns to calculating the line losses.

### Loss Calculations for Short Line Model

In the Short Line Model, the line is represented by its series impedance only. As a result, loss calculations include only series resistance and inductive reactance. With the load current  $I_{\text{load\_mag}}$  already determined, the active and reactive losses were then calculated. The following code illustrates this calculation:

```
active_losses_kw_short = (I_load_mag ** 2) * resistance * 3 / 1000 # kW
```

```
reactive_losses_kvar_short = (I_load_mag ** 2) * inductance * 3 / 1000 # kVAR
```

## Loss Calculations for Medium Line Model

In the medium line model, the transmission line is represented by a  $\pi$  model that includes both series impedance and shunt admittance. First, the total series impedance  $z$  and shunt admittance  $y$  per unit length are defined. The load is modeled as a complex power  $S_{\text{load}}$ , and the phase voltage  $V_{\text{rated}}$  and load current  $I_{\text{load}}$  are computed. The  $\pi$ -model then calculates the capacitive currents at the receiving and sending ends ( $I_{c2}$  and  $I_{c1}$ ), the current through the series impedance ( $I_1$ ), and the sending-end voltage  $V_{\text{sending}}$ . Finally, the active losses are determined from the magnitude of  $I_1$  and the series resistance, while the reactive losses account for the difference between inductive and capacitive contributions. The following code illustrates this calculation:

```
z = resistance + 1j * reactance # Total series impedance
y = 1j * susceptance # Total shunt admittance

S_load = complex(P_input * 1000, Q_input * 1000)
V_rated = V / math.sqrt(3) # Phase voltage (line-to-neutral)
I_load = np.conj(S_load / (V_rated * 3))

#  $\pi$ -model calculations
I_c2 = V_rated * y / 2 # Capacitive current at receiving end
I_1 = I_load + I_c2 # Current through series impedance
V_sending = V_rated + I_1 * z # Sending end voltage (complex)
I_c1 = V_sending * y / 2 # Capacitive current at sending end
I_sending = I_1 + I_c1

# Losses
active_losses_kw_medium = (abs(I_1) ** 2) * resistance * 3 / 1000
Q_L = (abs(I_1) ** 2) * reactance * 3 / 1000
Q_C = ((abs(V_sending) ** 2) * (susceptance / 2)
       + (abs(V_rated) ** 2) * (susceptance / 2)) * 3 / 1000
reactive_losses_kvar_medium = Q_L - Q_C
```

## Voltage Regulation Calculation

The percentage voltage regulation is calculated by combining the voltage drops caused by series resistance and inductance and then normalizing by the square of the line-to-line voltage. The following code illustrates this calculation:

```
voltage_regulation = 1000 * (P_input * resistance + Q_input * inductance) * 100 / (V **
2)
```

## Economic Analysis Calculations

Firstly, the installation cost is calculated from the unit price per kilometer in the cable specifications dataset. For three-core cables, the cost is equal to the price per kilometer multiplied by the length of the cable in kilometers and the number of parallel circuits. For single-core cables, this result is further multiplied by three to account for the three conductors required per phase. After that, it is given that a unit electricity price of 2500 TL per MWh. Daily operating hours are assigned based on the load type: 10 hours for industrial, 5 hours for residential, 12 hours for municipal, and 8 hours for commercial loads. The annual energy loss in kilowatt-hours is then computed by multiplying the active loss in kilowatts by the hours per day and by 365 days. Dividing this result by 1000 converts kWh to MWh, which is then multiplied by the electricity price to obtain the annual loss cost in Turkish Lira. Finally, the total loss cost over a ten-year period is estimated by multiplying the annual cost by 10. Only active power losses were monetized in these calculations; no cost was attributed to reactive power, so the economic analysis remains the same for both the Short and Medium line models. The following code illustrates this calculation:

```
electricity_price = 2500 # TL per MWh
hours_per_day = {
    'Industrial': 10,
    'Residential': 5,
    'Municipal': 12,
    'Commercial': 8
}[load_type]

annual_energy_loss_kwh = active_losses_kw_medium * hours_per_day * 365
annual_loss_cost_medium = (annual_energy_loss_kwh / 1000) * electricity_price
total_loss_cost_10yrs_medium = annual_loss_cost_medium * 10 # TL
```

## 4 Test Results

To validate the accuracy and usability of the application, three different test scenarios were conducted. Each scenario includes a unique load case and environmental setup. The figures below illustrate the input parameters, resulting cable selection, and final calculation outputs.

**Example 1:** A 2000 kW / 1500 kVAR industrial load at 5 kV with 30°C environment was tested using three-core cables. The application identified cable ID 28 as the most efficient. The medium-line model resulted in lower reactive losses and a total 10-year cost reduction of nearly 100,000 TL compared to the short-line model.

**Example 2:** For a high-power industrial load of 72 MW / 30 MVAR at 33 kV with 10°C ambient temperature, a single-core trefoil configuration was tested. The selected cable minimized voltage regulation (4.02%) and the economic analysis showed a 10-year energy loss of approximately 13.8 million TL using the medium-line model.

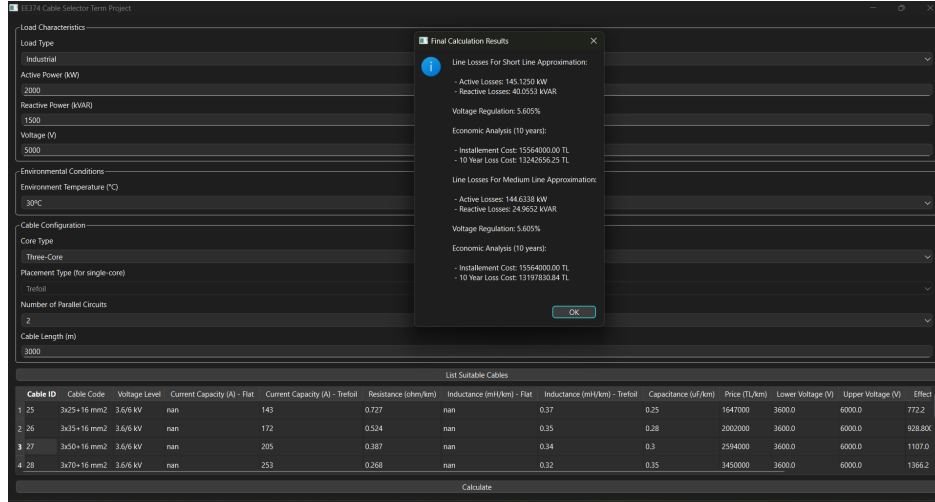


Figure 1. Example 1 – Cable Selection for 5 kV Industrial Load

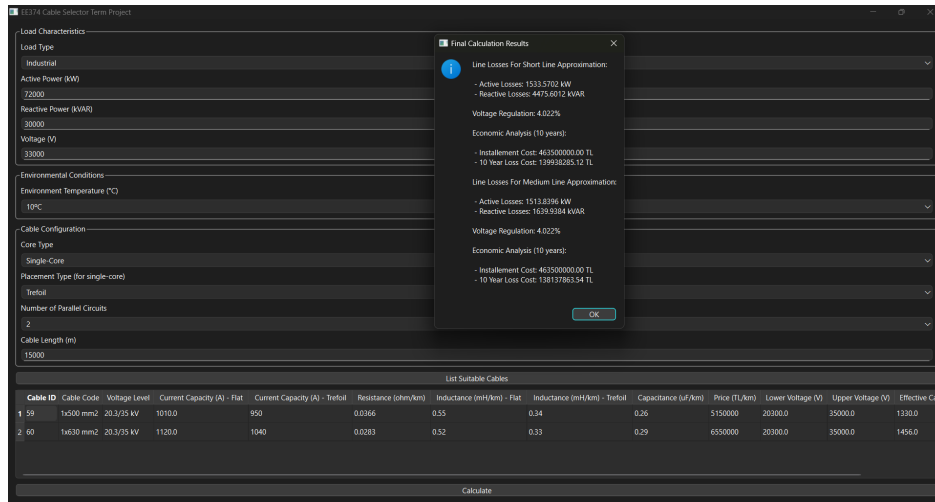


Figure 2. Example 2 – Cable Selection for 33 kV High Power Load

**Example 3:** A low-voltage test was performed for a 400 kW / 300 kVAR load at 800 V with 20°C ambient temperature. The tool identified multiple valid cable options. Losses were minimal (15 kW), and voltage regulation was below 3.2%. Both short and medium models gave nearly identical results due to the short cable length (400 m).

These test cases confirm the program's ability to handle varying voltage levels, power ratings, and cable types, while consistently producing accurate electrical and economic analyses.

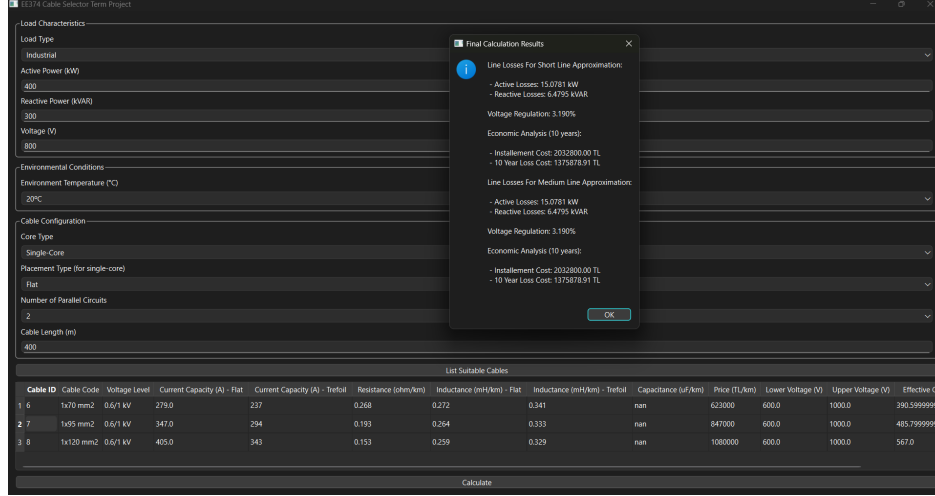


Figure 3. Example 3 – Cable Selection for 800 V Small Load

## 5 Discussion

The cable selection tool developed in this project successfully meets its intended goals of technical accuracy, operational safety, and economic feasibility.

The Smart Listing algorithm has proven effective in narrowing down cable options based on both thermal and electrical constraints. By integrating temperature and trench correction factors directly into the current capacity evaluation, the tool ensures compliance with real-world installation standards. The layered filtering approach—first by voltage rating, then by core configuration, and finally by depleted current capacity provides a logical selection path, while the GUI’s feedback mechanisms promote user awareness and prevent misconfiguration.

The voltage regulation and loss analysis modules demonstrated high reliability during testing. By offering both short and medium transmission line models, the application allows users to assess the impact of capacitive charging currents and line impedance under varying conditions. This dual modeling approach gives engineers the flexibility to choose between computational simplicity and modeling accuracy based on line length and required precision.

The economic analysis is quantifying installation cost and long-term energy losses in monetary terms. Using predefined energy prices and load type-dependent operation durations, the program provides an insightful ten-year cost comparison.

Overall, this project provides a solid foundation for automatic cable selection in power systems by combining an interface with accurate engineering calculations in a Python-based application.