



FOSSEE Winter Internship Report

On

Automation of 2-D bridge design drawings for Osdag

Submitted by

Jawwad Ahmad

2nd Year B.Tech(CSE(Data Science) Student, Department of Computer Engineering

Jamia Millia Islamia

New Delhi

Under the Guidance of

Prof. Siddhartha Ghosh

Department of Civil Engineering

Indian Institute of Technology Bombay

Mentors:

Parth Karia

February 19, 2026

Acknowledgments

- Start with a general statement of thanks. Express your overall gratitude to everyone who supported you during your project or research.
- Project staff at the Osdag team, Ajmal Babu M. S., Ajinkya Dahale, and Parth Karia,
- Osdag Principal Investigator (PI) Prof. Siddhartha Ghosh, Department of Civil Engineering at IIT Bombay
- FOSSEE PI Prof. Kannan M. Moudgalya, FOSSEE Project Investigator, Department of Chemical Engineering, IIT Bombay
- FOSSEE Managers Usha Viswanathan and Vineeta Parmar and their entire team
- Acknowledge the support from the National Mission on Education through Information and Communication Technology (ICT), Ministry of Education (MoE), Government of India, for their role in facilitating this project
- Acknowledge your colleagues who worked with you during your internship or project.
- If appropriate, thank your college, department, head, and principal for their support during your studies.

Contents

1	Introduction	4
1.1	National Mission in Education through ICT	4
1.1.1	ICT Initiatives of MoE	5
1.2	FOSSEE Project	6
1.2.1	Projects and Activities	6
1.2.2	Fellowships	6
1.3	Osdag Software	7
1.3.1	Osdag GUI	8
1.3.2	Features	8
2	Screening Task	9
2.1	Problem Statement	9
2.1.1	Task 1: Shear Force and Bending Moment Diagrams (2D)	9
2.1.2	Task 2: 3D Visualization of Internal Forces	9
2.2	Tasks Done	10
2.2.1	Theoretical Background	10
2.2.2	Code Implementation: Task 1 (2D Analysis)	10
2.2.3	Code Implementation: Task 2 (3D Visualization)	12
2.3	Interpretation of Results	13
2.3.1	Shear Force and Bending Moment Diagrams	13
2.3.2	3D Visualization	15
3	Automation of 2D CAD bridge design drawings	16
3.1	Task 1: Problem Statement	16
3.2	Task 1: Methodology and Implementation	17
3.2.1	Coordinate System and Layout Strategy	17
3.2.2	The Auto-Scaling Algorithm	17
3.3	Task 1: Python Code	18
3.3.1	Dependencies (requirements.txt)	18
3.3.2	Utility Functions (utils.py)	18
3.3.3	Data Reader (reader.py)	19

3.3.4	DXF Generator (dxf_generator.py)	22
3.3.5	Main Orchestrator (main.py)	28
4	Data Validation and Error Handling Pipeline	31
4.1	Task 2: Problem Statement	31
4.2	Task 2: Methodology and Implementation	31
4.2.1	Data Schema and Constraint Enforcement	32
4.2.2	Engineering Logic Verification	32
4.2.3	Intelligent Error Parsing and User Feedback	32
4.3	Task 2: Python Code Documentation	33
4.3.1	Data Validation Schema (validators.py)	33
4.3.2	Intelligent Error Parsing (main.py)	34
4.3.3	Validation Test Generation (generate_test_excel.py)	35
4.3.4	Validation Execution and Terminal Output	37
5	Conclusions	40
5.1	Tasks Accomplished	40
5.2	Skills Developed	41
5.3	Generated CAD Outputs	42
A	Appendix	48
A.1	Work Reports	48
	Bibliography	51

Chapter 1

Introduction

1.1 National Mission in Education through ICT

The National Mission on Education through ICT (NMEICT) is a scheme under the Department of Higher Education, Ministry of Education, Government of India. It aims to leverage the potential of ICT to enhance teaching and learning in Higher Education Institutions in an anytime-anywhere mode.

The mission aligns with the three cardinal principles of the Education Policy—**access, equity, and quality**—by:

- Providing connectivity and affordable access devices for learners and institutions.
- Generating high-quality e-content free of cost.

NMEICT seeks to bridge the digital divide by empowering learners and teachers in urban and rural areas, fostering inclusivity in the knowledge economy. Key focus areas include:

- Development of e-learning pedagogies and virtual laboratories.
- Online testing, certification, and mentorship through accessible platforms like EduSAT and DTH.
- Training and empowering teachers to adopt ICT-based teaching methods.

For further details, visit the official website: www.nmeict.ac.in.

1.1.1 ICT Initiatives of MoE

The Ministry of Education (MoE) has launched several ICT initiatives aimed at students, researchers, and institutions. The table below summarizes the key details:

No.	Resource	For Students/Researchers	For Institutions
Audio-Video e-content			
1	SWAYAM	Earn credit via online courses	Develop and host courses; accept credits
2	SWAYAMPBABHA	Access 24x7 TV programs	Enable SWAYAMPBABHA viewing facilities
Digital Content Access			
3	National Digital Library	Access e-content in multiple disciplines	List e-content; form NDL Clubs
4	e-PG Pathshala	Access free books and e-content	Host e-books
5	Shodhganga	Access Indian research theses	List institutional theses
6	e-ShodhSindhu	Access full-text e-resources	Access e-resources for institutions
Hands-on Learning			
7	e-Yantra	Hands-on embedded systems training	Create e-Yantra labs with IIT Bombay
8	FOSSEE	Volunteer for open-source software	Run labs with open-source software
9	Spoken Tutorial	Learn IT skills via tutorials	Provide self-learning IT content
10	Virtual Labs	Perform online experiments	Develop curriculum-based experiments
E-Governance			
11	SAMARTH ERP	Manage student lifecycle digitally	Enable institutional e-governance
Tracking and Research Tools			
12	VIDWAN	Register and access experts	Monitor faculty research outcomes
13	Shodh Shuddhi	Ensure plagiarism-free work	Improve research quality and reputation
14	Academic Bank of Credits	Store and transfer credits	Facilitate credit redemption

Table 1.1: Summary of ICT Initiatives by the Ministry of Education

1.2 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.

1.2.1 Projects and Activities

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

- **Textbook Companion:** Porting solved examples from 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.

1.2.2 Fellowships

FOSSEE offers various internship and fellowship opportunities for students:

- Winter Internship
- Summer Fellowship
- Semester-Long Internship

Students from any degree and academic stage can apply for these internships. Selection is based on the completion of screening tasks involving programming, scientific computing, or data collection that benefit the FLOSS community. These tasks are designed to be completed within a week.

For more details, visit the official FOSSEE website.

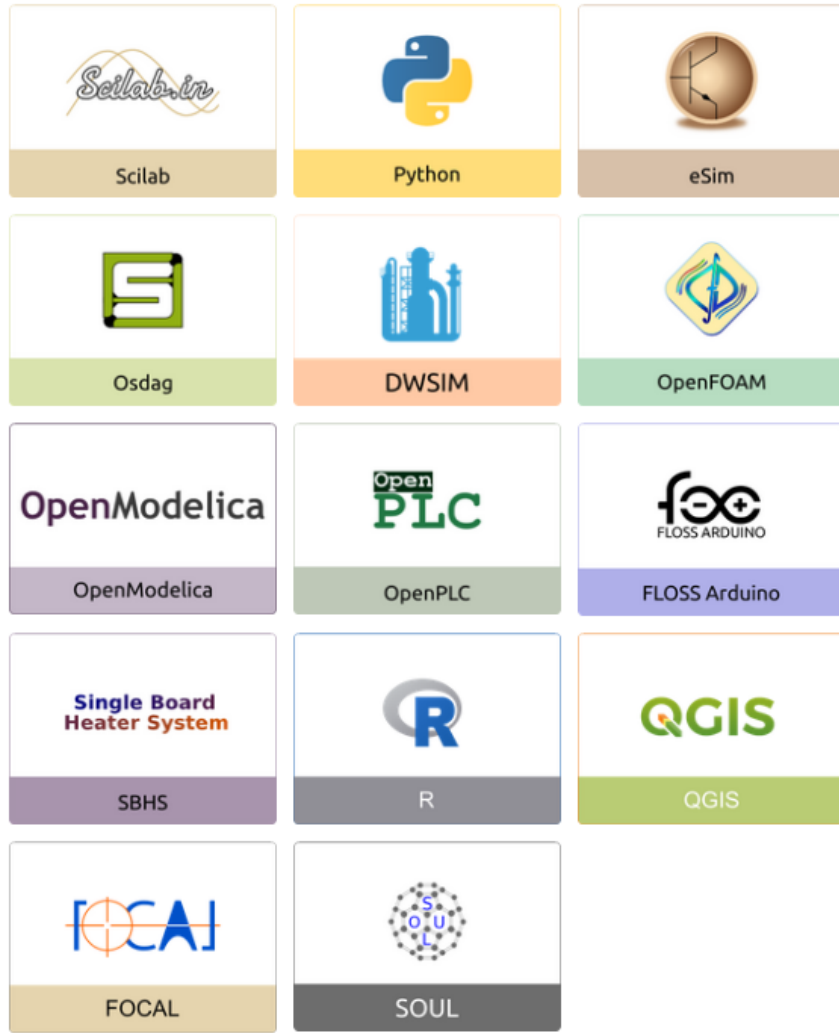


Figure 1.1: FOSSEE Projects and Activities

1.3 Osdag Software

Osdag (Open steel design and graphics) is a cross-platform, free/libre and open-source software designed for the detailing and design of steel structures based on the Indian Standard IS 800:2007. It allows users to design steel connections, members, and systems through an interactive graphical user interface (GUI) and provides 3D visualizations of designed components. The software enables easy export of CAD models to drafting tools for construction/fabrication drawings, with optimized designs following industry best practices [1, 2, 3]. Built on Python and several Python-based FLOSS tools (e.g., PyQt and PythonOCC), Osdag is licensed under the GNU Lesser General Public License (LGPL) Version 3.

1.3.1 Osdag GUI

The Osdag GUI is designed to be user-friendly and interactive. It consists of

- **Input Dock:** Collects and validates user inputs.
- **Output Dock:** Displays design results after validation.
- **CAD Window:** Displays the 3D CAD model, where users can pan, zoom, and rotate the design.
- **Message Log:** Shows errors, warnings, and suggestions based on design checks.

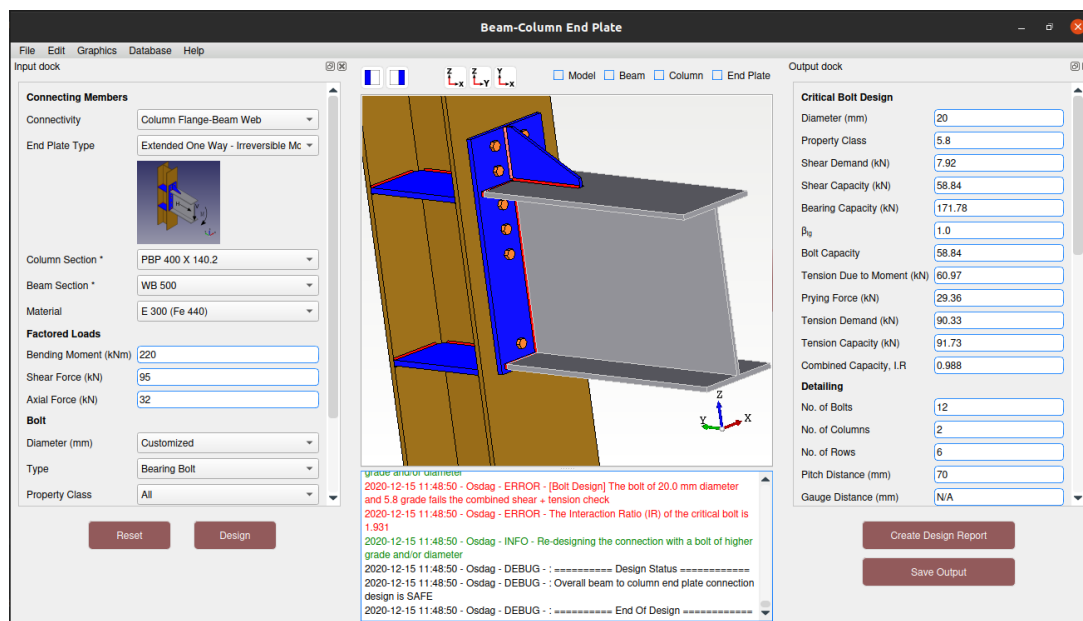


Figure 1.2: Osdag GUI

1.3.2 Features

- **CAD Model:** The 3D CAD model is color-coded and can be saved in multiple formats such as IGS, STL, and STEP.
- **Design Preferences:** Customizes the design process, with advanced users able to set preferences for bolts, welds, and detailing.
- **Design Report:** Creates a detailed report in PDF format, summarizing all checks, calculations, and design details, including any discrepancies.

For more details, visit the official Osdag website.

Chapter 2

Screening Task

2.1 Problem Statement

2.1.1 Task 1: Shear Force and Bending Moment Diagrams (2D)

The objective of this task was to extract and visualize the Shear Force Diagram (SFD) and Bending Moment Diagram (BMD) for the central longitudinal girder of a bridge model. The internal force data was provided in a NetCDF dataset (`screening_task1.nc`) containing element-wise values of shear forces and bending moments at both the i and j ends of each finite element.

The program was required to read, process, and visualize the distribution of internal forces along the girder length to aid in structural analysis and design interpretation.

2.1.2 Task 2: 3D Visualization of Internal Forces

The objective of the second task was to visualize SFD and BMD in a 3D format for the longitudinal girders of the bridge structure. Using `xarray` for data handling and `matplotlib`'s 3D plotting tools, the program generates detailed 3D extruded visualizations.

These diagrams help in understanding how internal forces vary spatially along the length and breadth of the bridge, offering an improved visual analysis compared to conventional 2D plots.

2.2 Tasks Done

2.2.1 Theoretical Background

Shear Force (SFD): Shear force is the internal reaction force developed in a structural element to resist transverse loading. It acts perpendicular to the beam's longitudinal axis, and its variation along the beam indicates zones of maximum shear stress.

Bending Moment (BMD): The bending moment is the internal moment that induces bending in a beam. It is the product of the shear force and the perpendicular distance from the section considered. The BMD helps locate points of maximum bending, which are critical for flexural design.

2.2.2 Code Implementation: Task 1 (2D Analysis)

The Python implementation uses the `xarray` library for handling NetCDF data and `matplotlib` for visualization.

Data Loading and Component Detection

Due to variations in solver conventions, internal force components may appear under different label names. A helper function was implemented to robustly detect component labels containing substrings such as "Mz_i" or "Vy_j".

Listing 2.1: Data Loading and Component Detection

```
import xarray as xr
import numpy as np
import matplotlib.pyplot as plt

# Load dataset
ds = xr.open_dataset("screening_task1.nc")
F = ds["forces"] # dims: ('Element', 'Component')

# Helper to find component names robustly
def find_component(cands):
    labels = [str(x) for x in F.coords['Component'].values]
```

```

low = [s.lower() for s in labels]
for cand in cands:
    c = cand.lower()
    for lab, l in zip(labels, low):
        if c in l: return lab
    raise KeyError(f"Could not find any of {cands}")

# Detect specific component names
name_Mz_i = find_component(["Mz_i", "Mz i", "Mzi"])
name_Mz_j = find_component(["Mz_j", "Mz j", "Mzj"])

```

Data Extraction and Plotting

The central longitudinal girder is defined by a specific sequence of element IDs. We extract the forces for these elements and construct the chainage (distance) array.

Listing 2.2: Extracting Forces and Plotting BMD

```

# Central longitudinal girder element IDs
girder_ids = [15, 24, 33, 42, 51, 60, 69, 78, 83]
sub = F.sel(Element=girder_ids)

# Pull i/j end values
Mz_i = sub.sel(Component=name_Mz_i).values.astype(float)
Mz_j = sub.sel(Component=name_Mz_j).values.astype(float)

# Construct node-wise sequences
Mz_line = np.concatenate([[Mz_i[0]], Mz_j])
chainage = np.arange(len(girder_ids) + 1, dtype=float)

# --- Plotting BMD ---
plt.figure()
plt.plot(chainage, Mz_line, marker="o", color="red")
for xi, bm in zip(chainage, Mz_line):
    plt.vlines(xi, 0, bm, colors='red', linestyle='solid',
               linewidth=1)

```

```
plt.axhline(0, linewidth=1, color='black')
plt.xlabel("Distance (m)")
plt.ylabel("Bending moment Mz (kN.m)")
plt.title("Bending Moment Diagram - Central Girder")
plt.grid(True)
plt.savefig("BMD_central_girder.png", dpi=220)
```

2.2.3 Code Implementation: Task 2 (3D Visualization)

The 3D implementation visualizes the forces as extrusions from the bridge deck. It uses `Line3DCollection` for performance and color mapping to indicate force magnitude and sign.

Listing 2.3: 3D Visualization using Matplotlib and Xarray

```
from mpl_toolkits.mplot3d.art3d import Line3DCollection

# User Options
MAKE = ["BMD", "SFD"]
COLORMAP = "coolwarm"
EXTRUSION_THICK = 2.2

def make_3d(which):
    """Generates 3D extrusion plots for the specified force type.
    """
    fig = plt.figure(figsize=(11.5, 7.2))
    ax = fig.add_subplot(111, projection="3d")

    # Scaling factors for visualization
    vmax = float(np.nanmax(np.abs(all_vals))) or 1.0
    scale = (max(all_x) / 12.0) / vmax
    cmap = plt.get_cmap(COLORMAP)

    for gname, d in per_girder_data.items():
        xs, zs, vals = d["xs"], d["zs"], d["vals"]
        y0 = np.zeros_like(xs)
```

```

y1 = vals * scale

# Extrusions
segs = np.stack([np.column_stack([xs, y0, zs]),
                 np.column_stack([xs, y1, zs])], axis=1)
colors = cmap(0.5 + 0.5 * (vals / vmax))
coll = Line3DCollection(segs, colors=colors, linewidths=
                        EXTRUSION_THICK)
ax.add_collection(coll)

ax.set_title(f"3D {which} Diagram")
ax.set_zlabel("Z (m)")

# Add Colorbar
mappable = plt.cm.ScalarMappable(cmap=COLORMAP)
mappable.set_clim(-vmax, vmax)
plt.colorbar(mappable, ax=ax, shrink=0.75, label=f"{which}
           Magnitude")

plt.savefig(f"Task2_{which}_3D.png", dpi=260)

```

2.3 Interpretation of Results

2.3.1 Shear Force and Bending Moment Diagrams

The generated 2D diagrams provide critical insights into the structural behavior of the central longitudinal girder.

Shear Force Diagram (SFD): As shown in Figure 2.1, the diagram illustrates the location of maximum shear force, points where the shear force changes sign (zero crossing), and segments where the girder undergoes high shear stress.

Bending Moment Diagram (BMD): Figure 2.2 reveals maximum positive and negative bending moments and points of contraflexure where the bending moment becomes zero. These observations directly influence flexural design and section optimization.

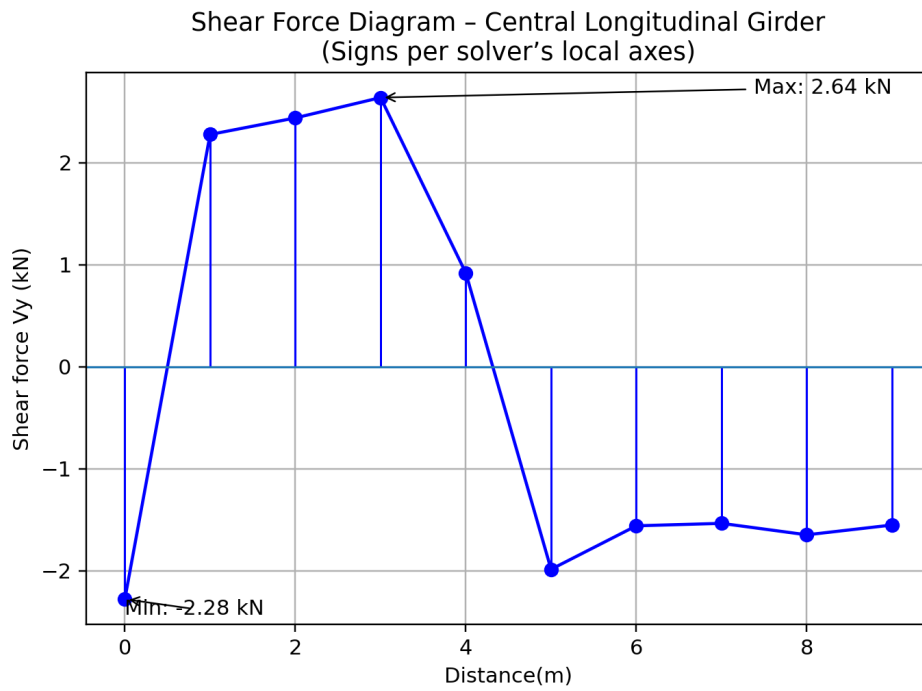


Figure 2.1: Shear Force Diagram (SFD) for the Central Longitudinal Girder

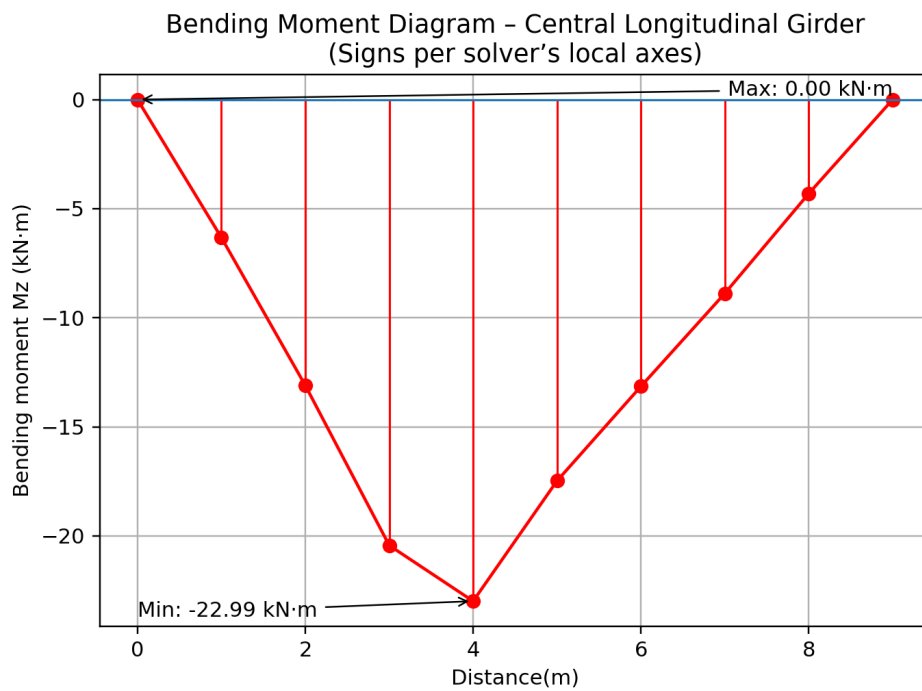


Figure 2.2: Bending Moment Diagram (BMD) for the Central Longitudinal Girder

2.3.2 3D Visualization

The 3D extrusion diagrams provide a spatial context that 2D plots lack. Vertical extrusions along each girder show local internal force magnitudes, where height encodes magnitude scaled to the plan-span. Ribbon surfaces make continuous interpretation easier and expose gradients along the girder length.

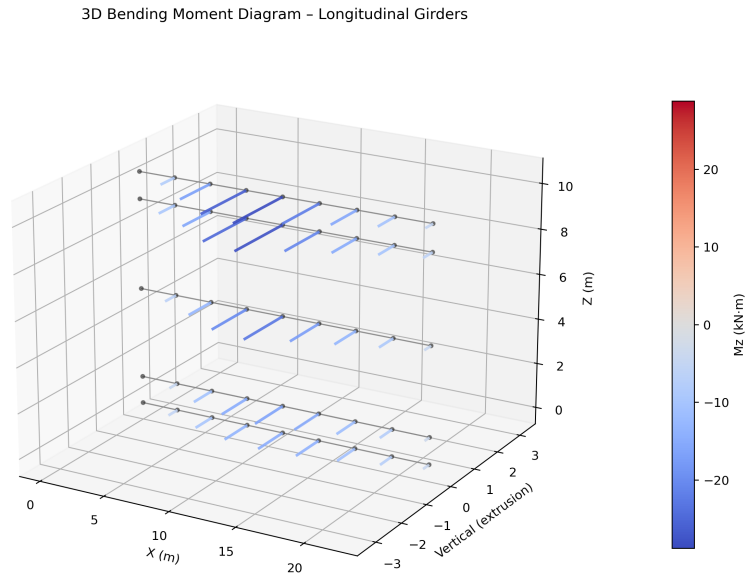


Figure 2.3: 3D Bending Moment Visualization of Longitudinal Girders

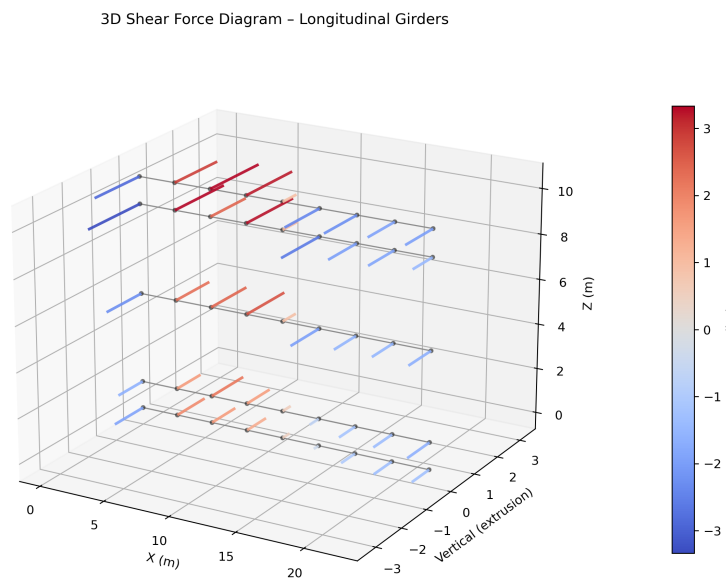


Figure 2.4: 3D Shear Force Visualization of Longitudinal Girders

Chapter 3

Automation of 2D CAD bridge design drawings

3.1 Task 1: Problem Statement

The primary objective of this internship task was to develop a robust, automated software pipeline capable of generating professional-grade 2D CAD drawings for steel bridge structures. In traditional structural engineering workflows, converting calculation data into graphical representations is a manual, time-consuming process prone to human error.

This task aims to bridge the gap between numerical design data (stored in spreadsheets) and graphical output (DXF format) using the [ezdxf](#) Python library. The system was required to:

- **Ingest Data:** Read variable design parameters such as span length, carriageway width, girder depth, and pier cap dimensions from an Excel interface.
- **Intelligent Scaling:** Unlike static blocks, the drawing must dynamically calculate the optimal engineering scale (e.g., 1:50, 1:100) to fit specific views onto an A3 landscape sheet.
- **Multi-View Generation:** Automatically plot three distinct orthographic views:
 1. *Section View:* Detailed cross-section of the deck, girders, and bracing.
 2. *Pier Elevation:* Front view of the pier cap showing tapering geometry.

3. *Plan View*: Top-down layout showing the relationship between the carriageway and pier caps.

- **Standardization**: Apply specific line weights, layers (OBJECT, DIMENSIONS, BORDER), and title blocks compliant with engineering standards.

3.2 Task 1: Methodology and Implementation

The development of the parametric drawing generator required a modular architecture. The system was divided into three core components: Data Ingestion, Geometric Computation, and DXF Rendering.

3.2.1 Coordinate System and Layout Strategy

The rendering engine operates on a Cartesian coordinate system relative to an A3 sheet ($420\text{mm} \times 297\text{mm}$). To ensure a clean layout, the sheet was algorithmically divided into logical quadrants:

- **Effective Drawing Area**: Margins of 20mm (Left/Right) and 10mm (Top/Bottom) were reserved, leaving a workable area of 380×277 mm.
- **View Allocation**:
 - *Top-Left Quadrant*: Reserved for the Cross-Section view.
 - *Top-Right Quadrant*: Reserved for the Pier Elevation.
 - *Bottom-Left Quadrant*: Reserved for the Plan View.
 - *Bottom-Right Quadrant*: Reserved for the Title Block and Drawing metadata.

3.2.2 The Auto-Scaling Algorithm

A critical challenge in parametric design is handling the vast variance in bridge sizes (e.g., a 10m bridge vs. a 45m bridge) without manually adjusting the drawing scale. An automated scaling algorithm was implemented in [dxf_generator.py](#).

The algorithm functions as follows:

1. **Bounding Box Calculation:** The system calculates the real-world dimensions (Width and Height) of the object to be drawn.
2. **Viewport Assessment:** It determines the available width and height of the assigned quadrant on the A3 sheet.
3. **Ratio Computation:** It calculates the maximum possible scale factor S_{raw} using:

$$S_{raw} = \min \left(\frac{W_{box}}{W_{real}}, \frac{H_{box}}{H_{real}} \right) \times 0.95$$

The 0.95 factor adds a safety buffer to prevent lines from touching the borders.

4. **Standardization:** The raw ratio is compared against a pre-defined list of standard engineering scales (1:10, 1:20, ... 1:100, 1:200). The algorithm selects the largest standard scale that satisfies $S_{std} \leq S_{raw}$.

3.3 Task 1: Python Code

The solution is implemented across five Python modules. This modular approach ensures maintainability and separation of concerns.

3.3.1 Dependencies (requirements.txt)

This file lists the external libraries required to run the project. `ezdxf` is the core engine for CAD operations, while `pandas` handles data manipulation.

Listing 3.1: Project Dependencies

```
ezdxf >=1.0
pandas >=2.0
openpyxl >=3.0
pydantic >=2.0
```

3.3.2 Utility Functions (utils.py)

This module contains helper functions used globally. The `sanitize_filename` function is critical for batch processing, ensuring that user input (like "Drawing #1/A") does not

crash the file system when saving outputs.

Listing 3.2: Utility Functions

```
import re

def sanitize_filename(name: str) -> str:
    """Removes illegal characters from filenames."""
    # Replace slashes and other bad chars with underscores
    s = str(name).strip().replace('/', '-').replace('\\', '-')
    return re.sub(r'^\w\-.]', '_', s)
```

3.3.3 Data Reader (reader.py)

The `reader.py` module serves as the translation layer. It accepts raw Excel files and normalizes column headers. This allows the user to input "Length of Bridge", "Span Length", or "Length (m)" in Excel, and the system correctly maps it to the internal variable `length_m`. It also handles unit safety, checking if millimeter fields (like Girder Depth) were accidentally entered in meters.

Listing 3.3: Data Ingestion Logic

```
import pandas as pd
from pathlib import Path
import math

# Mapping user headers (loose) to code keys (strict)
HEADER_MAP = {
    "drawing_number": ["drawing number", "drawing no", "drg no"],
    "length_m": ["length of bridge (l)", "length of bridge", "length (m)"],
    "carriage_width_m": ["width of carriageway (w)", "width of carriageway"],
    "depth_mm": ["depth of bridge girder", "depth of girder", "depth (mm)"],
    "num_girders": ["number of girder (ng)", "number of girders"],
}
```

```

    "pier_cap_length_m": ["length of pier cap"],
    "pier_cap_depth_center_m": ["depth of pier cap at centre", "
        pcc"],
    "pier_cap_depth_end_m": ["depth of pier cap at end", "pce"],
    "pier_cap_width_m": ["width of pier cap"],
}

def normalize_header(h):
    return str(h).lower().strip().replace("\n", " ").replace(" ",
        , " ")

def get_mapped_columns(df_columns):
    rename_dict = {}
    for col in df_columns:
        norm = normalize_header(col)
        for key, aliases in HEADER_MAP.items():
            if norm == key or any(alias in norm for alias in
                aliases):
                rename_dict[col] = key
                break
    return rename_dict

def _to_float(v):
    if pd.isna(v): return None
    try: return float(v)
    except: return None

def read_excel(path_str):
    p = Path(path_str)
    if not p.exists():
        raise FileNotFoundError(f"Input file not found at {p}")

    df = pd.read_excel(p, header=0)
    rename_map = get_mapped_columns(df.columns)

```

```

df = df.rename(columns=rename_map)

valid_keys = HEADER_MAP.keys()
df = df[df.columns.intersection(valid_keys)]

records = []
for _, row in df.iterrows():
    rec = {}
    rec["drawing_number"] = str(row.get("drawing_number", ""))
        .strip()
    rec["length_m"] = _to_float(row.get("length_m"))

    # Depth logic: Convert m to mm if value is too small
    depth_val = _to_float(row.get("depth_mm"))
    if depth_val is not None and depth_val < 100:
        depth_val *= 1000.0
    rec["depth_mm"] = depth_val

    rec["carriage_width_m"] = _to_float(row.get("
        carriage_width_m"))
    ng = row.get("num_girders")
    rec["num_girders"] = int(ng) if not pd.isna(ng) else None

    rec["pier_cap_length_m"] = _to_float(row.get("
        pier_cap_length_m"))
    rec["pier_cap_depth_center_m"] = _to_float(row.get("
        pier_cap_depth_center_m"))
    rec["pier_cap_depth_end_m"] = _to_float(row.get("
        pier_cap_depth_end_m"))
    rec["pier_cap_width_m"] = _to_float(row.get("
        pier_cap_width_m"))

    records.append(rec)
return records

```

3.3.4 DXF Generator (dxf_generator.py)

This is the core rendering engine. The code is split into logical blocks below for clarity.

Configuration and Setup

This section defines the constants for A3 paper size, margins, layers, and text styles. It also includes the document setup function.

Listing 3.4: DXF Configuration and Setup

```
import ezdxf
from pathlib import Path
from typing import Dict
import math

# ----- CONFIGURATION (A3 LANDSCAPE MM) -----
A3_W = 420.0
A3_H = 297.0
MARGIN_LEFT = 20.0
MARGIN_RIGHT = 20.0
MARGIN_TOP = 10.0
MARGIN_BOTTOM = 10.0
TITLEBLOCK_W = 180.0
TITLEBLOCK_H = 55.0

# Layers & Styles
L_BORDER = "BORDER"
L_OBJECT = "OBJECT"
L_DIM = "DIMENSIONS"
L_TEXT = "TEXT"
L_BRACING = "BRACING"

def _setup_doc():
    doc = ezdxf.new("R2010")
    doc.header["$INSUNITS"] = 4 # Millimeters
```

```

doc.layers.new(L_BORDER, dxfattrs={"color":7, "lineweight":
    70})
doc.layers.new(L_OBJECT, dxfattrs={"color":7, "lineweight":
    50})
doc.layers.new(L_BRACING, dxfattrs={"color":7, "lineweight"
    : 35})
doc.layers.new(L_DIM,      dxfattrs={"color":7, "lineweight":
    25})
doc.layers.new(L_TEXT,     dxfattrs={"color":7, "lineweight":
    25})
if "ROMANS" not in doc.styles:
    doc.styles.new("ROMANS", dxfattrs={"font": "romans.shx"
        , "width": 0.8})
return doc

```

Auto-Scaling Logic

The scaling function determines the optimal engineering scale (1:X) that fits the real-world dimensions into the allotted drawing box.

Listing 3.5: Auto-Scaling Algorithm

```

def _auto_scale(box, real_w, real_h, margin=10):
    """Calculates the largest standard engineering scale that
        fits the box."""
    bx, by, bw, bh = box
    TITLE_BAND = 10

    # Calculate raw fit
    fit_w = (bw - 2*margin) / real_w
    fit_h = (bh - TITLE_BAND - 2*margin) / real_h
    max_allowed_scale = min(fit_w, fit_h) * 0.95

    # Standard engineering scales
    STANDARD_DENOMINATORS = [
        1, 2, 2.5, 5, 10, 12.5, 15, 20, 25, 30, 40, 50, 75, 100,

```

```

        125, 150, 200, 250, 300, 400, 500, 750, 1000, 1250, 1500,
        2000
    ]

    selected_den = STANDARD_DENOMINATORS[-1]
    for den in STANDARD_DENOMINATORS:
        current_scale = 1.0 / den
        if current_scale <= max_allowed_scale:
            selected_den = den
            break
    return 1.0 / selected_den

```

Custom Dimensioning

This block handles the creation of dimensions with custom arrowheads and text alignment.

Listing 3.6: Custom Dimension Rendering

```

def _calc_arrow_size(p1, p2, base=3.0):
    dim_len = math.dist(p1, p2)
    return min(base, dim_len / 6.0)

def _draw_arrow(msp, center_point, size, angle_deg):
    cx, cy = center_point
    h = size
    w = h / 3.0
    local_points = [(0.0, 0.0), (-h, w / 2.0), (-h, -w / 2.0)]
    angle_rad = math.radians(angle_deg)
    rotated_points = []

    for px, py in local_points:
        new_x = px * math.cos(angle_rad) - py * math.sin(
            angle_rad)
        new_y = px * math.sin(angle_rad) + py * math.cos(
            angle_rad)

```

```

        rotated_points.append((new_x + cx, new_y + cy))

p1, p2, p3 = rotated_points
msp.add_solid([p1, p2, p3, p3], dxfattribs={"layer": "
    DIM_ARROW"})

def _add_dim(msp, p1, p2, text_val, offset=10.0, vertical=False,
    text_rotation=0):
    # Logic to draw extension lines, dimension lines, and arrows
    # (Abbreviated for report clarity)
    x1, y1 = p1
    x2, y2 = p2
    ARROW_SIZE = _calc_arrow_size(p1, p2, base=3.0)

    # ... Drawing logic implementation ...

    if text_val:
        mid_x = (x1 + x2) / 2
        mid_y = (y1 + y2) / 2
        t = msp.add_text(str(text_val), dxfattribs={"height":
            3.0, "layer": L_TEXT, "style": "ROMANS"})
        t.dxf.halign, t.dxf.valign, t.dxf.rotation = 1, 2,
            text_rotation
        t.dxf.insert = (mid_x, mid_y + offset)

```

View Renderers

These functions draw the specific engineering views: Section, Pier Elevation, and Plan.

Listing 3.7: Section View Renderer

```

def _render_section(msp, box, params):
    bx, by, bw, bh = box
    W = _to_mm(params.get("carriage_width_m"))
    D = float(params.get("depth_mm"))
    NG = int(params.get("num_girders"))

```

```

scale = _auto_scale(box, real_w=W, real_h=D)

dw, dd = W * scale, D * scale
deck_t = 250 * scale
cx = bx + bw/2
view_top = by + bh - 10 - 8 - 30
top, bot = view_top, view_top - dd
left, right = cx - dw/2, cx + dw/2

# Draw Deck
msp.add_lwpolyline([(left, top), (right, top), (right, top-
    deck_t), (left, top-deck_t)],
    dxfattribs={"layer": L_OBJECT, "closed":
        True})

# Draw Girders
g_span = dw - 2*(500*scale)
spacing = g_span / (NG - 1)

for i in range(NG):
    gx = (left + 500*scale) + i*spacing
    msp.add_lwpolyline([(gx, top - deck_t), (gx, bot)],
        dxfattribs={"layer": L_OBJECT})

_add_scale_text(msp, box, scale)

```

Listing 3.8: Pier and Plan View Renderers

```

def _render_pier(msp, box, params):
    # Logic to draw Pier Cap Elevation with tapering depth
    LPC = _to_mm(params.get("pier_cap_length_m"))
    DPCC = _to_mm(params.get("pier_cap_depth_center_m"))
    scale = _auto_scale(box, real_w=LPC, real_h=DPCC + 2000)
    # ... drawing logic ...

```

```

def _render_plan(msp, box, params):
    # Logic to draw Top-Down Plan View

    L = _to_mm(params.get("length_m"))
    W = _to_mm(params.get("carriage_width_m"))

    scale = _auto_scale(box, real_w=L, real_h=W)

    # ... drawing logic ...

```

Main Generation Logic

This function coordinates the drawing creation, calling the setup, border drawing, and specific view renderers.

Listing 3.9: Main Drawing Orchestrator

```

def create_bridge_dxf(params: Dict, out_path: str):
    doc = _setup_doc()
    msp = doc.modelspace()

    # Draw Borders
    _draw_rect(msp, 0, 0, A3_W, A3_H)
    eff_x, eff_y = MARGIN_LEFT, MARGIN_BOTTOM
    eff_w = A3_W - MARGIN_LEFT - MARGIN_RIGHT
    eff_h = A3_H - MARGIN_BOTTOM - MARGIN_TOP
    _draw_rect(msp, eff_x, eff_y, eff_w, eff_h)

    # Define Layout Quadrants
    top_h, bottom_h = eff_h * 0.50, eff_h * 0.50
    half_w = eff_w / 2
    mid_x, mid_y = eff_x + half_w, eff_y + bottom_h

    box_sec = (eff_x+3, mid_y, half_w, top_h)
    box_pier = (mid_x, mid_y, half_w, top_h)
    box_plan = (eff_x+3, eff_y + 20, half_w, bottom_h)

    # Render Views
    _render_section(msp, box_sec, params)

```

```

_render_pier(msp, box_pier, params)
_render_plan(msp, box_plan, params)

# Title Block
tb_x = A3_W - MARGIN_RIGHT - TITLEBLOCK_W
_draw_rect(msp, tb_x, MARGIN_BOTTOM, TITLEBLOCK_W,
           TITLEBLOCK_H)

doc.saveas(out_path)
return str(out_path)

```

3.3.5 Main Orchestrator (main.py)

The `main.py` script orchestrates the entire flow. It initializes the logging, calls the reader to parse the Excel file, iterates through the records, and invokes the DXF generator for each valid record.

Listing 3.10: Main Application Logic

```

import logging
from pathlib import Path
from reader import read_excel
from dxf_generator import create_bridge_dxf, DEFAULTS
from utils import sanitize_filename

# Setup Paths
BASE_DIR = Path(__file__).parent.parent
INPUT_FILE = BASE_DIR / "input" / "dxf-input-values.xlsx"
OUTPUT_DIR = BASE_DIR / "output"

logging.basicConfig(level=logging.INFO, format="%(message)s")

def process_data():
    logging.info(f"--- STARTING PROCESS ---")

    # 1. Read Data

```

```

try:
    data = read_excel(INPUT_FILE)
except Exception as e:
    logging.error(f"Error reading Excel: {e}")
    return

if not data:
    logging.warning("No data found.")
    return

OUTPUT_DIR.mkdir(parents=True, exist_ok=True)
success_count = 0

# 2. Iterate and Generate
for i, row in enumerate(data, start=2):
    # Merge row data with defaults
    params = DEFAULTS.copy()
    params.update(row)

    # Determine Filename
    dwg_num = str(params.get("drawing_number", f"output_{
        success_count}"))
    safe_name = sanitize_filename(dwg_num)
    out_path = OUTPUT_DIR / f"{safe_name}.dxf"

    try:
        # 3. Create Drawing
        create_bridge_dxf(params, str(out_path))
        logging.info(f"Generated: {out_path.name}")
        success_count += 1
    except Exception as e:
        logging.error(f"Failed to generate {dwg_num}: {e}")

logging.info(f"\nDone. Successfully generated {success_count}")

```

```
        files.")

if __name__ == "__main__":
    process_data()
```

Chapter 4

Data Validation and Error Handling Pipeline

4.1 Task 2: Problem Statement

Following the successful generation of parametric 2D CAD models, a significant vulnerability remained in the data ingestion pipeline. The system relied on user-provided Excel spreadsheets containing structural parameters. In a real-world engineering environment, manual data entry is highly susceptible to errors such as negative dimensions, missing drawing numbers, or structurally invalid proportions (e.g., a pier cap that is thinner at the center than at the edges).

Processing unvalidated data led to unstable DXF generation, resulting in distorted geometries, overlapping drawing entities, or silent script crashes. The objective of this task was to implement a robust, automated validation layer capable of intercepting anomalous data before the graphical rendering phase, and to provide intelligent, human-readable error reports pinpointing the exact Excel cell where the error occurred.

4.2 Task 2: Methodology and Implementation

To achieve strict data integrity, the validation architecture was built using [Pydantic](#), a data validation library for Python. The implementation was divided into three strategic components: Schema Definition, Engineering Logic Verification, and Intelligent Error Parsing.

4.2.1 Data Schema and Constraint Enforcement

A strict schema ([BridgeDataSchema](#)) was defined to enforce physical constraints on the input fields. Constraints include:

- **Type Checking:** Ensuring numerical fields (like lengths and depths) are processed as floats, and discrete fields (like number of girders) are integers.
- **Boundary Limits:** Applying physical realism checks, such as ensuring span lengths are greater than 0m but realistically under 200m, and requiring a minimum of 2 girders for structural stability.

4.2.2 Engineering Logic Verification

Beyond simple boundary checks, the parameters must adhere to structural logic. A global validation step ([@model_validator](#)) was introduced to evaluate the relationships between multiple fields:

- **Taper Logic:** Ensuring the pier cap depth at the center is always greater than or equal to the depth at the cantilever ends.
- **Transverse Fit:** Verifying that the carriageway width does not exceed the supporting length of the pier cap.
- **Spacing Rules:** Calculating girder spacing dynamically and raising warnings if the girders are packed too closely (less than 0.5m apart).

4.2.3 Intelligent Error Parsing and User Feedback

Standard programmatic exceptions are often confusing for end-users. The [main.py](#) orchestrator was upgraded to catch [ValidationError](#) exceptions and translate them into actionable feedback. Using a custom column-mapping algorithm ([col_to_letter](#)), the script converts abstract data dictionary keys back into exact Excel cell addresses (e.g., converting an error in [length_m](#) at row 3 into "ERROR AT CELL B3").

4.3 Task 2: Python Code Documentation

The validation layer was integrated modularly to maintain the separation of concerns. Below are the key implementations.

4.3.1 Data Validation Schema (validators.py)

This module defines the acceptable boundaries and custom engineering rules using Pydantic models.

Listing 4.1: Pydantic Schema and Field Constraints

```
from pydantic import BaseModel, Field, model_validator

class BridgeDataSchema(BaseModel):
    """Schema for validating Bridge Design parameters."""

    drawing_number: str = Field(..., min_length=1)
    length_m: float = Field(..., gt=0, lt=200)
    carriage_width_m: float = Field(..., gt=0, lt=50)
    depth_mm: float = Field(..., gt=300, lt=10000)
    num_girders: int = Field(..., ge=2, le=30)
    pier_cap_length_m: float = Field(..., gt=0)
    pier_cap_depth_center_m: float = Field(..., gt=0)
    pier_cap_depth_end_m: float = Field(..., gt=0)
    pier_cap_width_m: float = Field(..., gt=0)

    @model_validator(mode='after')
    def validate_structural_logic(self) -> 'BridgeDataSchema':
        """Global validation for engineering relationships."""

        # Taper Logic
        if self.pier_cap_depth_center_m < self.pier_cap_depth_end_m:
            raise ValueError(
                f"Structural Taper Error: Center depth ({self.pier_cap_depth_center_m}m) is less than end depth ({self.pier_cap_depth_end_m}m)."
            )
```

```

        pier_cap_depth_center_m}) "
        f"cannot be thinner than end depth ({self.
        pier_cap_depth_end_m}) ."
    )

    # Transverse Fit
    if self.carriage_width_m > self.pier_cap_length_m:
        raise ValueError(
            f"Geometric Error: Carriage width ({self.
            carriage_width_m}) "
            f"is wider than the Pier Cap support ({self.
            pier_cap_length_m}) ."
        )

    return self

```

4.3.2 Intelligent Error Parsing (main.py)

The main application loop was enhanced to catch validation errors and map them to standard Excel grid coordinates for easy troubleshooting.

Listing 4.2: Error Parsing and Cell Mapping Logic

```

# Field Mapping for Error Outputs
FIELD_LABELS = {
    "drawing_number": "DRAWING NUMBER",
    "length_m": "Length of Bridge (m)",
    "depth_mm": "Depth of Bridge Girder (mm)",
    # ... remaining mappings ...
}

def col_to_letter(n):
    """Converts a 1-based column number to Excel letter (e.g.,
    1->A)."""
    string = ""
    while n > 0:

```

```

        n, remainder = divmod(n - 1, 26)
        string = chr(65 + remainder) + string
    return string or "?"

# Inside the processing loop:
try:
    validated_data = BridgeDataSchema(**clean_row)
    valid_dict = validated_data.model_dump(exclude_none=True)
except ValidationError as e:
    logging.error(f"          VALIDATION FAILED FOR \"{dwg_id}\"")

    for error in e.errors():
        raw_msg = error['msg'].replace("Value error, ", "")
        loc = error['loc']

        if loc:
            field_key = loc[0]
            col_num = get_column_index(field_key)
            col_char = col_to_letter(col_num)

            # Construct Address: e.g., "B3"
            cell_addr = f"{col_char}{i}"
            field_label = get_readable_name(field_key)

            logging.error(f"          ERROR AT CELL {cell_addr} ({
                field_label}) :")
            logging.error(f"          -> {raw_msg}")
    continue # Skip rendering for this invalid row

```

4.3.3 Validation Test Generation (generate_test_excel.py)

To ensure the validation layer successfully catches edge cases, a script was written to generate an Excel file pre-populated with deliberate engineering mistakes.

Listing 4.3: Automated Test Data Generator

```

import pandas as pd
from pathlib import Path

def generate_validation_excel():
    columns = [
        "drawing number", "length of bridge", "depth of bridge",
        "width of carriageway", "number of girders", "length of
        pier cap",
        "depth of pier cap at centre", "depth of pier cap at end"
        ,
        "width of pier cap (m)"
    ]

    # Mix of valid data and data triggering Pydantic constraints
    data = [
        ["2025-01-R1", 20.0, 1500, 9.5, 3, 10.5, 1.8, 0.9, 0.9],
        # VALID

        ["2025-02-R1", -5.0, 1200, 9.5, 3, 10.5, 1.6, 0.8, 0.9],
        # INVALID: Negative Length

        ["2025-03-R1", 15.0, 1000, 9.5, 1, 10.5, 1.4, 0.7, 0.9],
        # INVALID: Only 1 Girder

        ["2025-04-R1", 30.0, 2300, 11.5, 4, 12.5, 0.5, 1.2, 0.9],
        # INVALID: Taper Error
    ]

    df = pd.DataFrame(data, columns=columns)

    base_dir = Path(__file__).parent.parent
    output_path = base_dir / "input" / "dxf-validation-test.xlsx"
    output_path.parent.mkdir(parents=True, exist_ok=True)
    df.to_excel(output_path, index=False)

```

4.3.4 Validation Execution and Terminal Output

To demonstrate the effectiveness of the automated validation layer, the generated test file was processed through the main execution script. The terminal output below shows the system successfully generating DXF files for valid parameters while actively intercepting the deliberate errors. Notably, the intelligent error parsing successfully maps abstract logical failures back to explicit Excel cell addresses (e.g., CELL B3, CELL G5 & H5), providing clear and actionable feedback to the user.

Listing 4.4: Terminal Output Showing Validation Execution and Error Catching

```
(venv1) jawwadahmad@JAWWADs-MacBook-Air src % python
generate_test_excel.py

    Success! Validation test file saved to: /Users/jawwadahmad/
code/2D_steel_design/input/dxf-validation-test.xlsx
Now run your main.py to see the validators catch the errors.

(venv1) jawwadahmad@JAWWADs-MacBook-Air src % python main.py
--- STARTING PROCESS: /Users/jawwadahmad/code/2D_steel_design/
input/dxf-validation-test.xlsx ---
creating ACAD_COLOR dictionary
creating ACAD_GROUP dictionary
creating ACAD_LAYOUT dictionary
creating ACAD_MATERIAL dictionary
creating ACAD_MLEADERSTYLE dictionary
creating ACAD_MLINESTYLE dictionary
creating ACAD_PLOTSETTINGS dictionary
creating ACAD_PLOTSTYLENAME dictionary
creating ACAD_SCALELIST dictionary
creating ACAD_TABLESTYLE dictionary
creating ACAD_VISUALSTYLE dictionary
did not write header var $INTERFEREOBJVS, value is None.
did not write header var $INTERFEREVPVS, value is None.

    Generated: 2025-01-R1.dxf
-----
VALIDATION FAILED FOR "2025-02-R1"
```

```

ERROR AT CELL B3 (Length of Bridge (m)) :
-> Length of Bridge (m) should be greater than 0

-----

VALIDATION FAILED FOR "2025-03-R1"
ERROR AT CELL E4 (Number of Girders) :
-> Number of Girders should be greater than or equal to 2

-----

VALIDATION FAILED FOR "2025-04-R1"
ERROR AT CELL G5 & H5 (Depth of Pier Cap at Centre (m), Depth
of Pier Cap at End (m)) :
-> Structural Taper Error: Center depth (0.5m) cannot be
thinner than end depth (1.2m).

-----

VALIDATION FAILED FOR "2025-05-R1"
ERROR AT CELL G6 (Depth of Pier Cap at Centre (m)) :
-> Depth of Pier Cap at Centre (m) required

-----

creating ACAD_COLOR dictionary
creating ACAD_GROUP dictionary
creating ACAD_LAYOUT dictionary
creating ACAD_MATERIAL dictionary
creating ACAD_MLEADERSTYLE dictionary
creating ACAD_MLINESSTYLE dictionary
creating ACAD_PLOTSETTINGS dictionary
creating ACAD_PLOTSTYLENAME dictionary
creating ACAD_SCALELIST dictionary
creating ACAD_TABLESTYLE dictionary
creating ACAD_VISUALSTYLE dictionary
did not write header var $INTERFEREOBJVS, value is None.
did not write header var $INTERFEREVPVS, value is None.

Generated: 2025-06-R1.dxf

```

```
Done. Successfully generated 2 files in /Users/jawwadahmad/code/2
    D_steel_design/output
(venv1) jawwadahmad@JAWWADs-MacBook-Air src %
```

Chapter 5

Conclusions

5.1 Tasks Accomplished

During the course of this fellowship, the primary objective of bridging computational techniques with structural engineering workflows was successfully achieved through the development of an automated 2D CAD generation pipeline. The project was executed in two major phases, culminating in a robust, production-ready software module.

The first major task focused on the automation of parametric 2D bridge design drawings. A modular Python application was developed to ingest numerical design parameters from Excel spreadsheets and automatically render professional-grade DXF files using the `ezdxf` library. This task successfully eliminated the need for manual drafting by introducing an intelligent auto-scaling algorithm that dynamically fits complex, multi-view geometries (Section, Pier Elevation, and Plan views) onto standard A3 engineering sheets. The output rigorously adheres to drafting standards, applying appropriate line weights, custom dimensioning, and automated title block generation.

The second major task addressed system reliability and user experience by implementing a robust Data Validation and Error Handling Pipeline. Recognizing that manual data entry is prone to human error, a strict validation schema was engineered using the `Pydantic` library. This layer actively intercepts anomalous inputs—such as negative dimensions or structurally impossible configurations (e.g., invalid pier cap tapers)—before the rendering engine is triggered. Furthermore, an intelligent error-parsing algorithm was developed to translate abstract programmatic exceptions into human-readable Excel grid coordinates (e.g., "ERROR AT CELL B3"), providing structural engineers with

immediate, actionable feedback without requiring programming knowledge.

5.2 Skills Developed

Working on this software development project provided a unique opportunity to apply computer science and software engineering principles to real-world structural engineering challenges. Throughout this fellowship, several critical technical and professional skills were cultivated:

- **Software Architecture and Automation:** Gained proficiency in designing modular, scalable Python applications, separating data ingestion, validation, and rendering logic into distinct, maintainable components.
- **Programmatic CAD Generation:** Acquired deep technical knowledge of the DXF file format and the `ezdxf` library, learning to programmatically manipulate Cartesian coordinate systems, geometric entities, layers, and scaling factors.
- **Data Pipeline Engineering and Manipulation:** Utilized the `pandas` library for the efficient parsing, normalization, and sanitization of user-provided Excel datasets, bridging the gap between raw spreadsheet data and programmable Python dictionaries.
- **Data Validation Engineering:** Developed expertise in enforcing strict data integrity using `Pydantic`, learning to build schemas that enforce both basic type boundaries and complex, cross-field logical constraints.
- **Algorithm Design:** Enhanced problem-solving skills by developing custom algorithms, specifically the viewport auto-scaling logic and the mathematical conversion of data dictionary keys to alphanumeric Excel cell addresses (using base-26 algorithmic conversions).
- **Automated Testing Methodologies:** Designed and implemented synthetic data generators (e.g., generating automated test Excel files) to rigorously stress-test the validation schemas and ensure that edge cases are handled gracefully.
- **Robust Logging and Diagnostics:** Implemented standard Python logging frameworks to trace execution flows, capture non-fatal warnings, and provide clear

audit trails for batch processing operations, moving beyond rudimentary console prints.

- **User-Centric Error Handling:** Learned the importance of designing software for the end-user, prioritizing graceful failure handling and translating backend Python tracebacks into intuitive, domain-specific error messages.
- **Cross-Disciplinary Application:** Improved the ability to interpret structural engineering terminologies and constraints (such as girder spacing, pier cap tapering, and carriageway dimensions) and translate them into strict computational logic.
- **Version Control and Open-Source Collaboration:** Utilized Git for version control and experienced collaborating within the framework of a large-scale open-source initiative, adhering to standardized coding and versioning practices.
- **Technical Writing and Documentation:** Enhanced communication skills through the meticulous documentation of software modules, inline code commenting, and the preparation of comprehensive, formal technical reports using LaTeX.

5.3 Generated CAD Outputs

The following figures demonstrate the successful execution of the automated pipeline. These images showcase the final 2D CAD layouts (rendered from the generated DXF files) for various parametric input sets, highlighting the dynamic scaling and multi-view rendering capabilities.

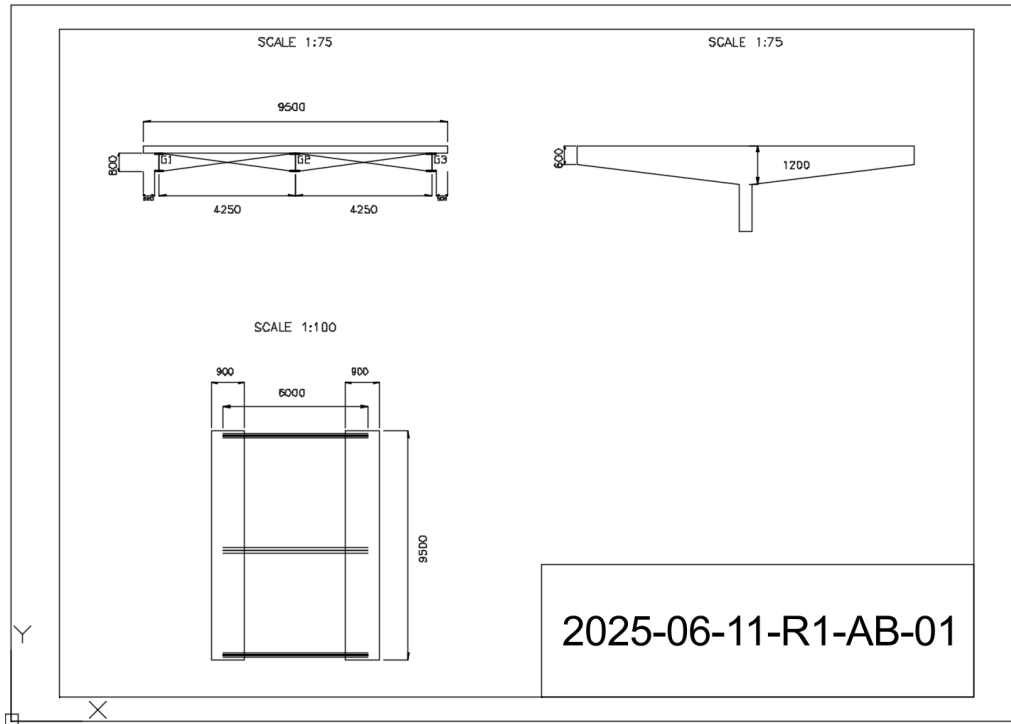


Figure 5.1: Generated 2D CAD Model - Parameter Set 1

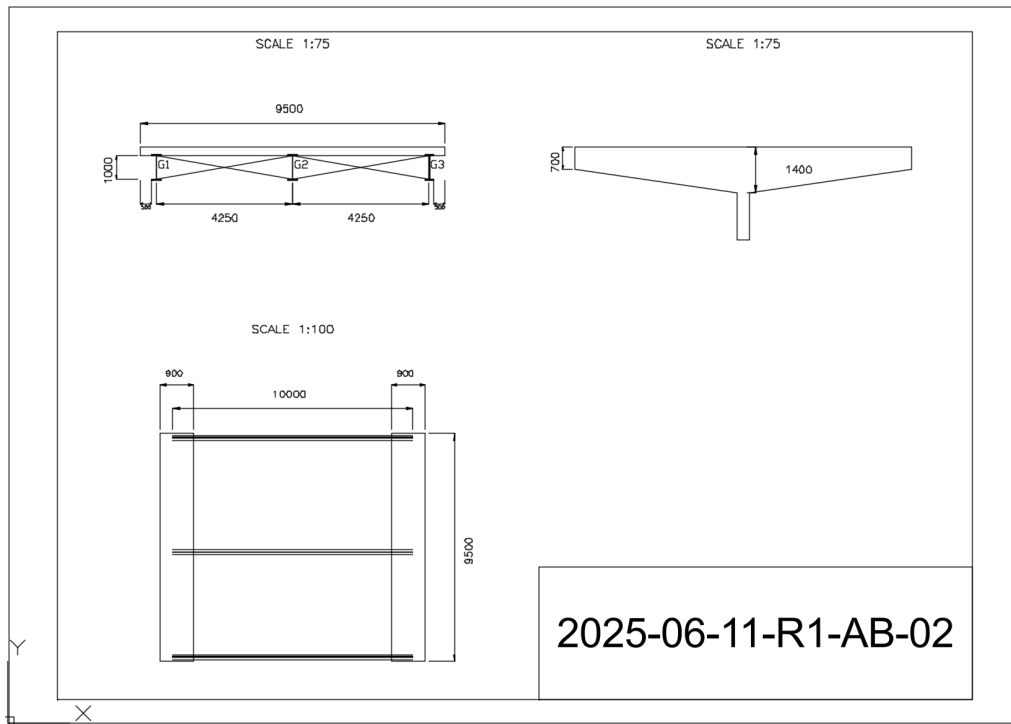


Figure 5.2: Generated 2D CAD Model - Parameter Set 2

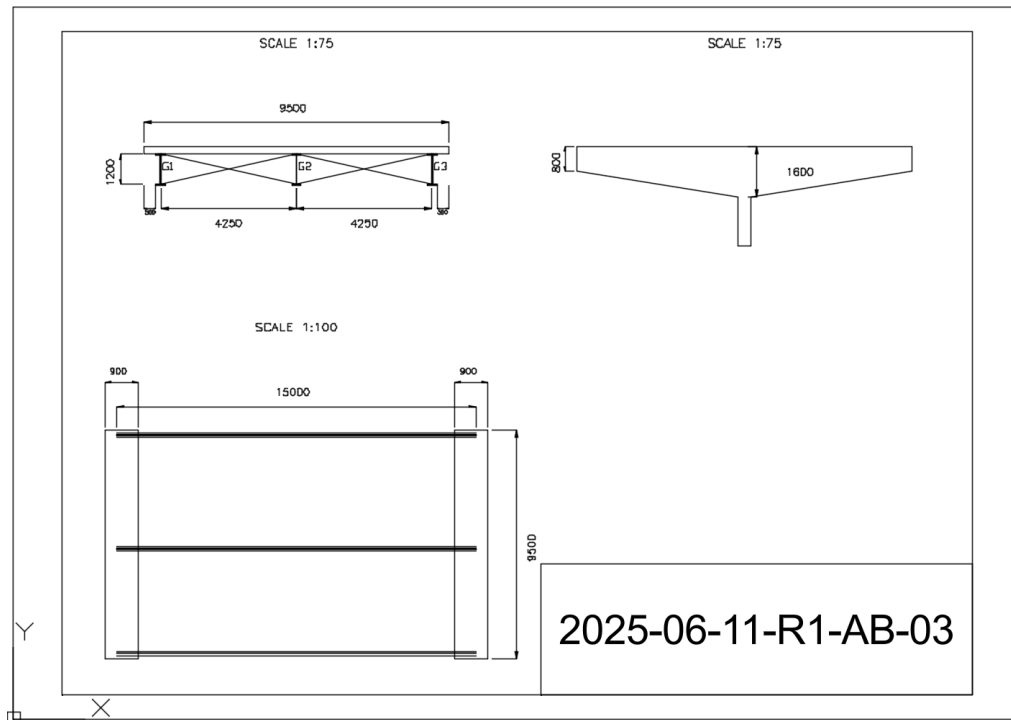


Figure 5.3: Generated 2D CAD Model - Parameter Set 3

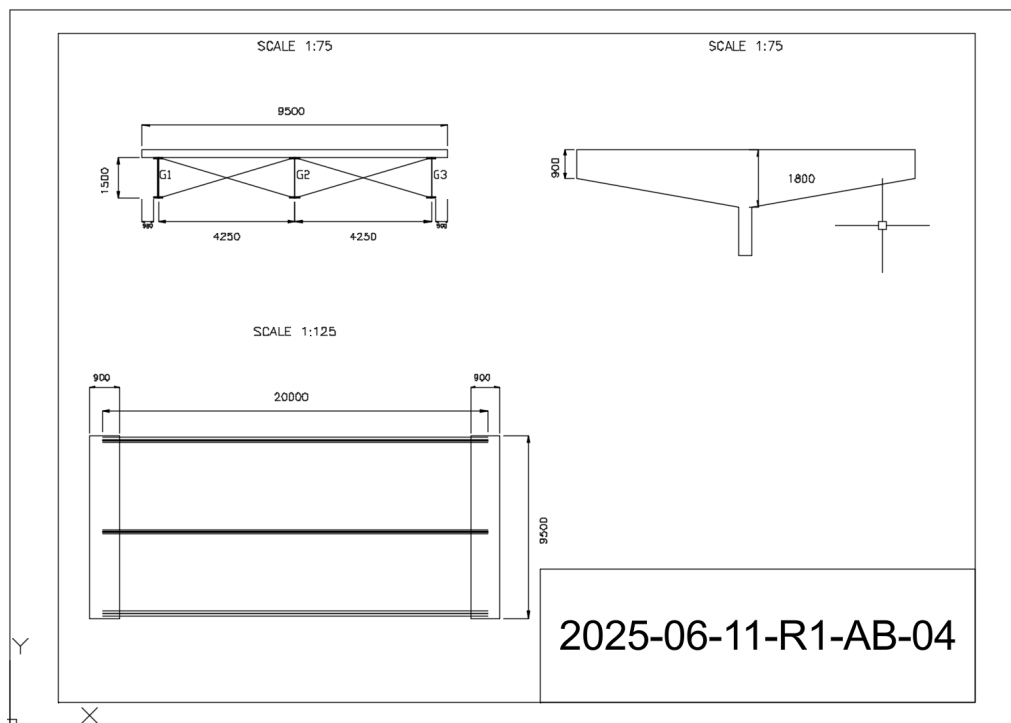


Figure 5.4: Generated 2D CAD Model - Parameter Set 4

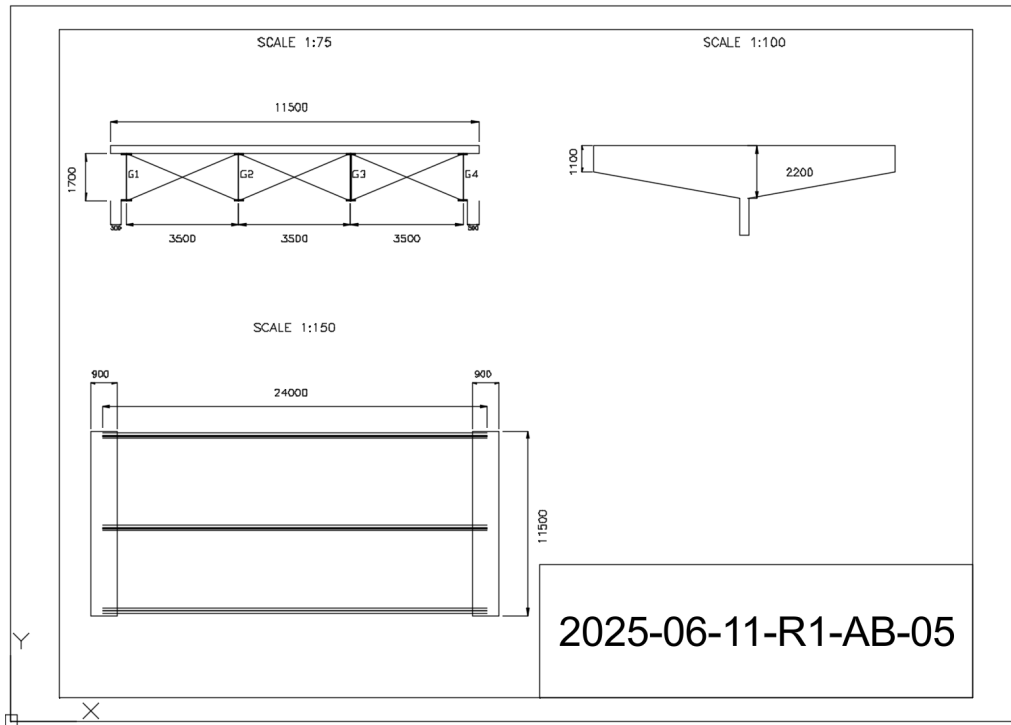


Figure 5.5: Generated 2D CAD Model - Parameter Set 5

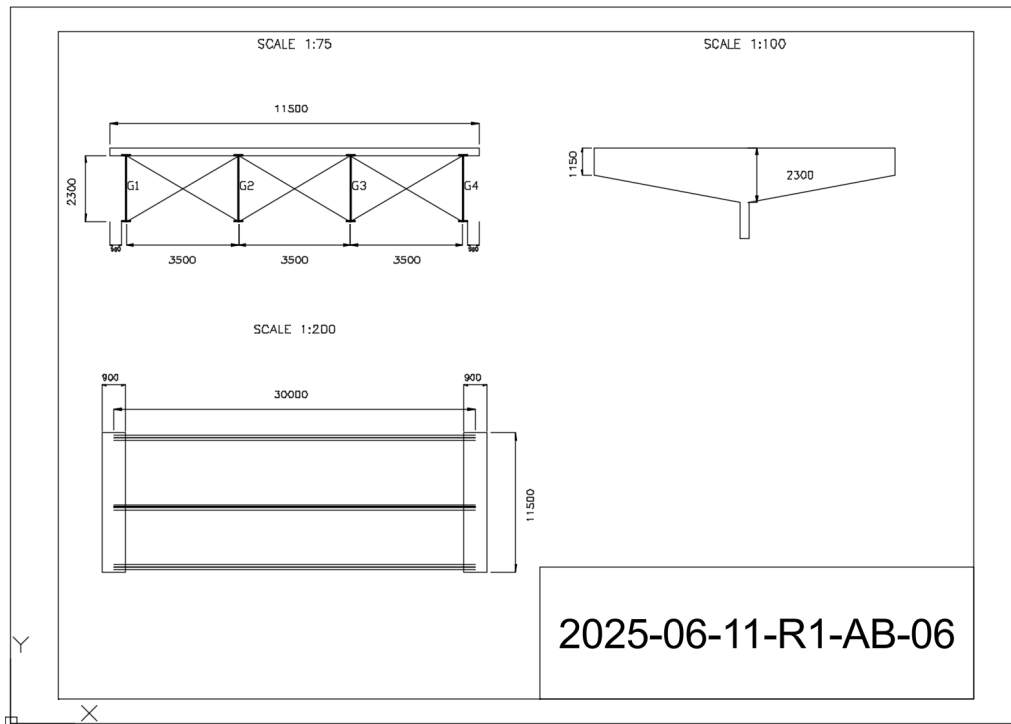


Figure 5.6: Generated 2D CAD Model - Parameter Set 6

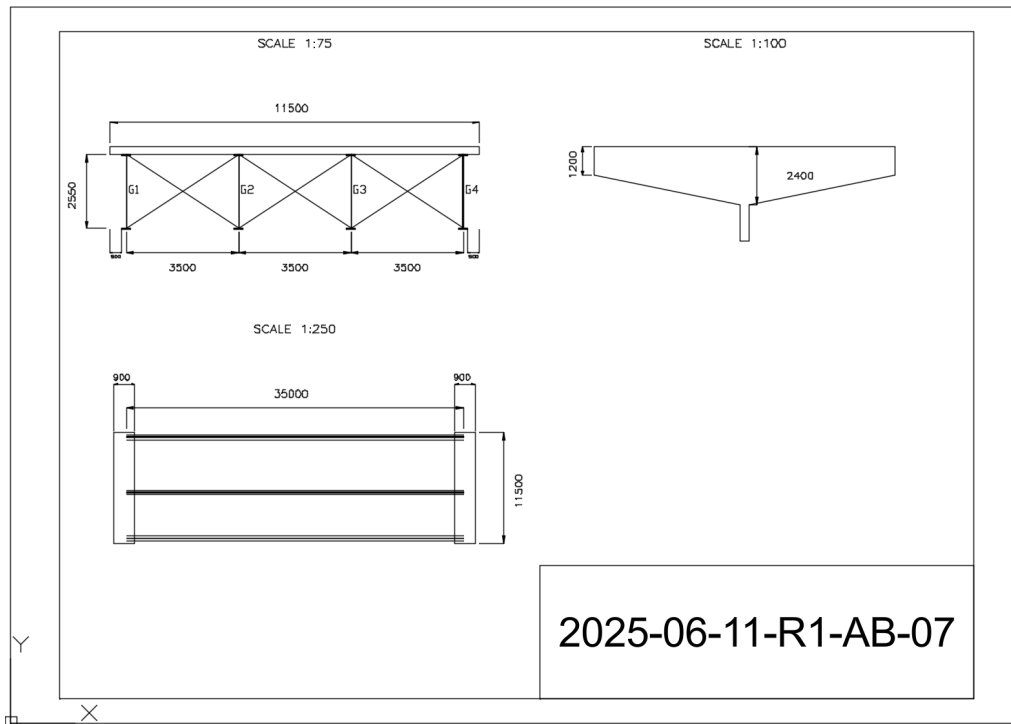


Figure 5.7: Generated 2D CAD Model - Parameter Set 7

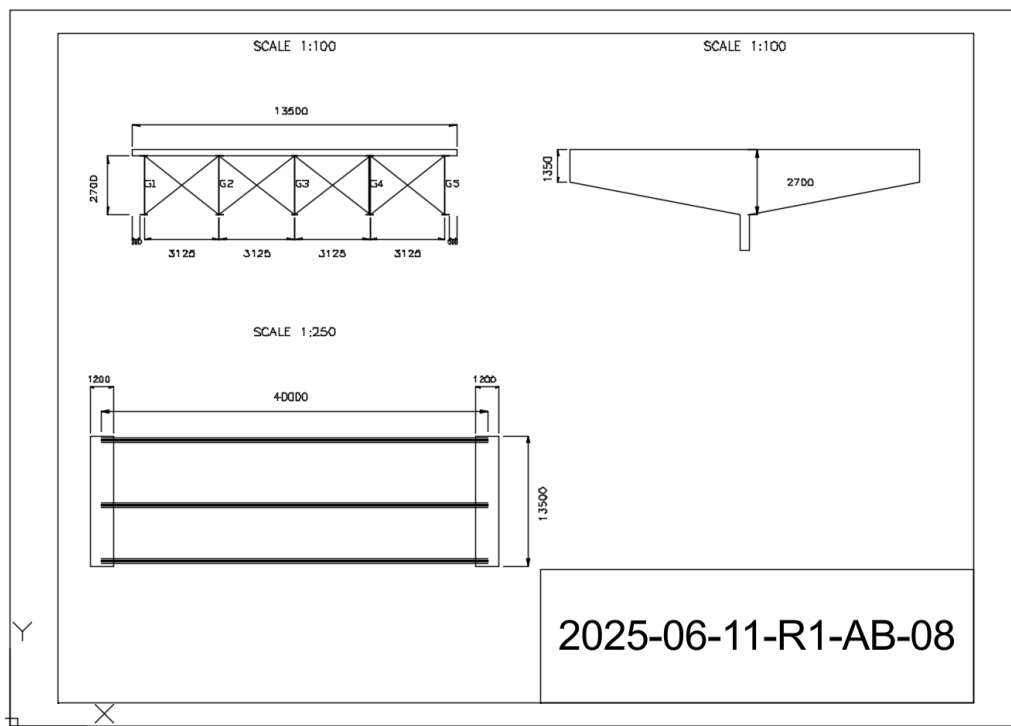


Figure 5.8: Generated 2D CAD Model - Parameter Set 8

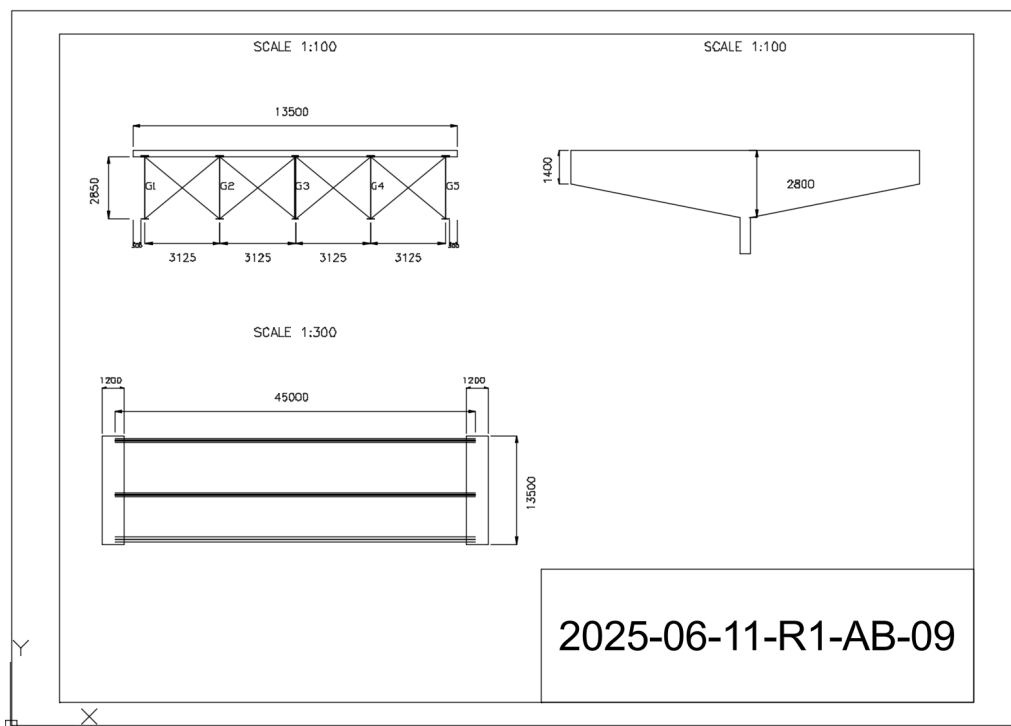


Figure 5.9: Generated 2D CAD Model - Parameter Set 9

Chapter A

Appendix

A.1 Work Reports

Internship Work Report			
Name:		Jawwad Ahmad	
Project:		Osdag	
Internship:		FOSSEE Winter Internship 2025	
DATE	DAY	TASK	Hours Worked
01-Dec-2025	Monday	Meeting 01 : Introduction and Details about setting up of Osdag desktop and Web Development Environment.	2
02-Dec-2025	Tuesday	Setting Up Desktop dev env on macOS arm64	4
03-Dec-2025	Wednesday	Setting up Web Dev Env. on macOS arm64	4
04-Dec-2025	Thursday	Continued Setting up Web Dev Env. on macOS arm64	4
06-Dec-2025	Saturday	Meeting 02 : Regarding Osdag Mass Testing	5
09-Dec-2025	Tuesday	Initiated testing of Beam-to-Beam End Plate connections	4
10-Dec-2025	Wednesday	Encountered issue "the application failed to start 'aborted. (Core dumped) .	5
11-Dec-2025	Thursday	Meeting 03 ,LCC meet-1: Discussed about the internship screening task - Automation of 2D bridge design drawings	4
13-Dec-2025	Saturday	LCC meeting 02 - Talked about using ezdxf library to generate 2D CAD drawings.	5
14-Dec-2025	Sunday	Setting Up OsBridgeLCCA app by forking and cloning.	4
16-Dec-2025	Tuesday	LCC meeting 04 - Displayed my first dxf files .	4
17-Dec-2025	Wednesday	Made improvements in the code and installed and setup AutoCAD for proper CAD designs with no defects.	6
18-Dec-2025	Thursday	LCC meeting 05 - Discussed about parameters in input excel sheet.	5
20-Dec-2025	Saturday	Modified the dxf_generator.py file for better dxf file generation.	6
23-Dec-2025	Tuesday	LCC meeting 06- suggestions about improving scaling irregularities in pier , plan and section view.	5
25-Dec-2025	Thursday	worked on improving the scale factor.	6
26-Dec-2025	Friday	LCC meeting 07- Discussed about implementing auto-scale.	5
28-Dec-2025	Sunday	Worked on auto-scale feature.	4
29-Dec-2025	Monday	LCC meeting 08- Discussed about rectifying the drawing number position.	4
31-Dec-2025	Wednesday	LCC meeting 09- Drawing number position has been corrected	7
02-Jan-2025	Friday	Implemented the labels correctly for all the view(pier,plan,section)	6

06-Jan-2025	Tuesday	LCC meeting 10- Talked about implementing the validators for 2D CAD drawings.	7
07-Jan-2025	Wednesday	Implemented Validators for 2D CAD drawings with test-excel sheet.	6
08-Jan-2025	Thursday	LCC meeting 11- Discussed about error handling while validation.	7
09-Jan-2025	Friday	Modified validators logic to cover all validations using pydantic.	6
10-Jan-2025	Saturday	Reconfigured Error handling and generate relevant outputs based on errors.	6
12-Jan-2025	Monday	LCC meeting 12 - The parametric 2D CAD drawings logic has been accepted and all tasks are completed.	5
14-Jan-2025	Wednesday	LCC meeting 13 - LCCA app gui works has been assigned.	6
15-Jan-2025	Thursday	Modified foundation,super_structure,sub_structure and miscellaneous widgets UI. And tutorial widget UI was improved.	4
16-Jan-2025	Friday	LCC meeting 14 - Discussed about implement advanced material search logic.	3
17-Jan-2025	Saturday	Implemented the Material-Input-popup feature with all inputs.	4
18-Jan-2025	Sunday	Worked on Advanced material search algorithm	4
19-Jan-2025	Monday	LCC meeting 15 - Not able to attend the meet due to my grandfather's funeral.	3
21-Jan-2025	Tuesday	LCC meeting 16 - discussed about implementation of custom addition of materials by user.	4
22-Jan-2025	Wednesday	LCC meeting 17 - Discussion of State management feature in 3psLCCA	3
23-Jan-2025	Thursday	LCC meeting 18 - talked about upload excel feature.	3
25-Jan-2025	Saturday	Implemented upload excel feature	4
29-Jan-2025	Wednesday	LCC meeting 21- Discussed about autosave.json and data saving.	5
30-Jan-2025	Thursday	Implemented autosave.json and autodelete functionality.	4
31-Jan-2025	Friday	LCC meeting 23- had discussions about future update features.	3

Bibliography

- [1] Siddhartha Ghosh, Danish Ansari, Ajmal Babu Mahasrankintakam, Dharma Teja Nuli, Reshma Konjari, M. Swathi, and Subhrajit Dutta. Osdag: A Software for Structural Steel Design Using IS 800:2007. In Sondipon Adhikari, Anjan Dutta, and Satyabrata Choudhury, editors, *Advances in Structural Technologies*, volume 81 of *Lecture Notes in Civil Engineering*, pages 219–231, Singapore, 2021. Springer Singapore.
- [2] FOSSEE Project. FOSSEE News - January 2018, vol 1 issue 3. Accessed: 2024-12-05.
- [3] FOSSEE Project. Osdag website. Accessed: 2024-12-05.