**FINAL REPORT**

**Civil Engineering Insight Studio**

**1. INTRODUCTION**

**1.1 Project Overview**

Civil Engineering Insight Studio is a web-based Generative AI application designed to automate structural analysis and documentation from construction site images. The system leverages Google Gemini Vision API to analyze images and generate structured engineering reports.

The application allows users to:

* Upload construction site images
* Enter structural analysis request
* Generate structured engineering report
* Identify materials and structural components
* View formatted AI-generated output

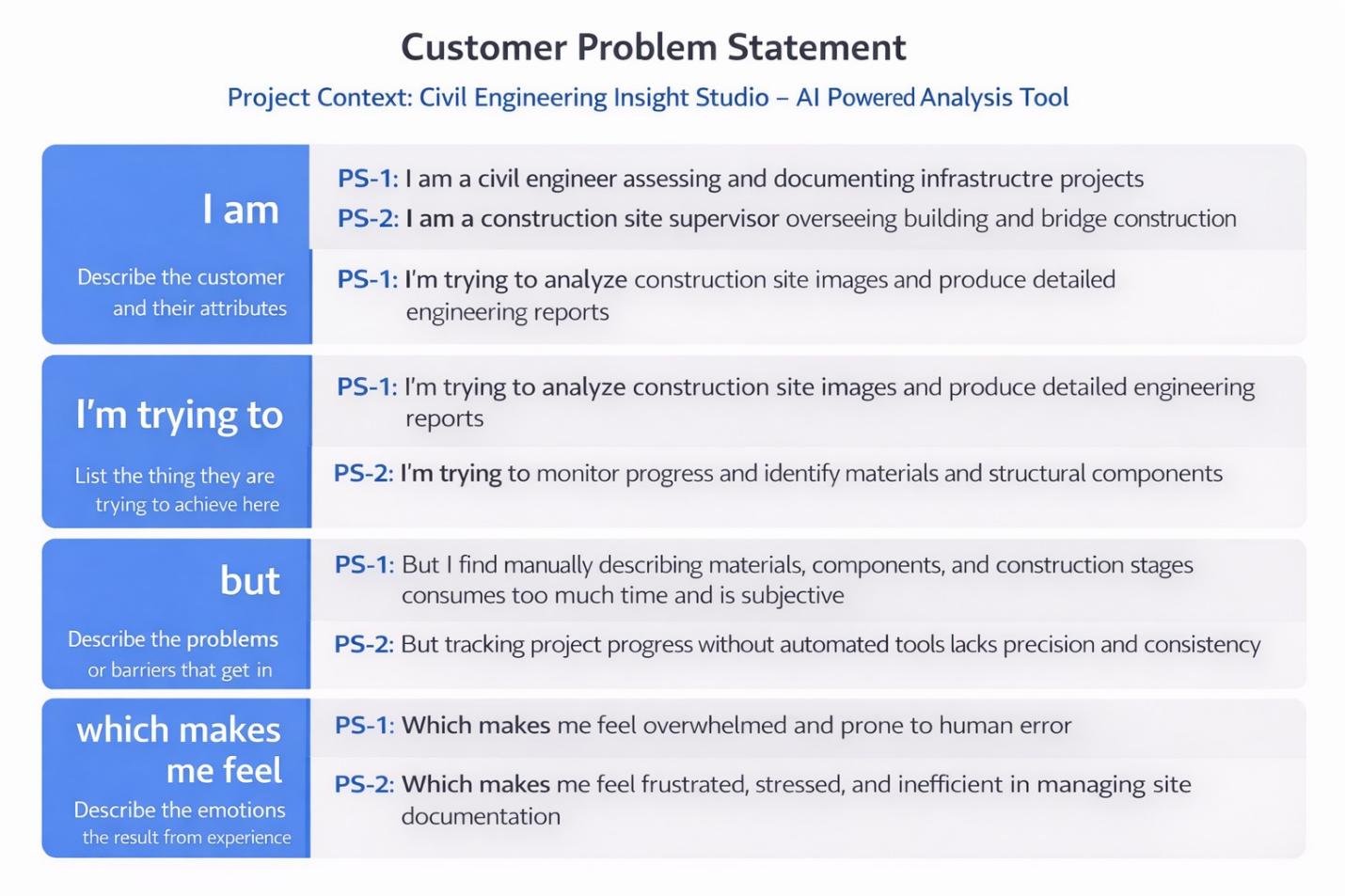
**1.2 Purpose**

* The purpose of this project is to reduce manual documentation effort in civil engineering projects, provide AI-assisted structural analysis, and demonstrate real-world integration of multimodal Generative AI within a deployable web application

**2. IDEATION PHASE**

**2.1 Problem Statement**

Civil engineers and construction supervisors spend significant time manually analyzing construction images and preparing structured reports. Traditional documentation methods are time-consuming and subjective.



**2.2 Empathy Map Canvas**

Target User: Civil Engineer / Site Supervisor  
  
Pain Points:  
• Time-consuming manual analysis  
• Risk of missing structural details  
• Inconsistent reporting format  
• Heavy workload under tight deadlines

**2.3 Brainstorming**

Among multiple AI-based ideas, the AI-powered civil structure analysis system was selected due to its strong industry relevance, technical feasibility, and clear problem–solution alignment.

Graphical user interface, application

Description automatically generated

**3. REQUIREMENT ANALYSIS**

### 3.1 Functional Requirements

* Accept construction image upload
* Accept structural analysis request
* Generate structured report via Gemini API
* Display formatted engineering output
* Handle API errors gracefully

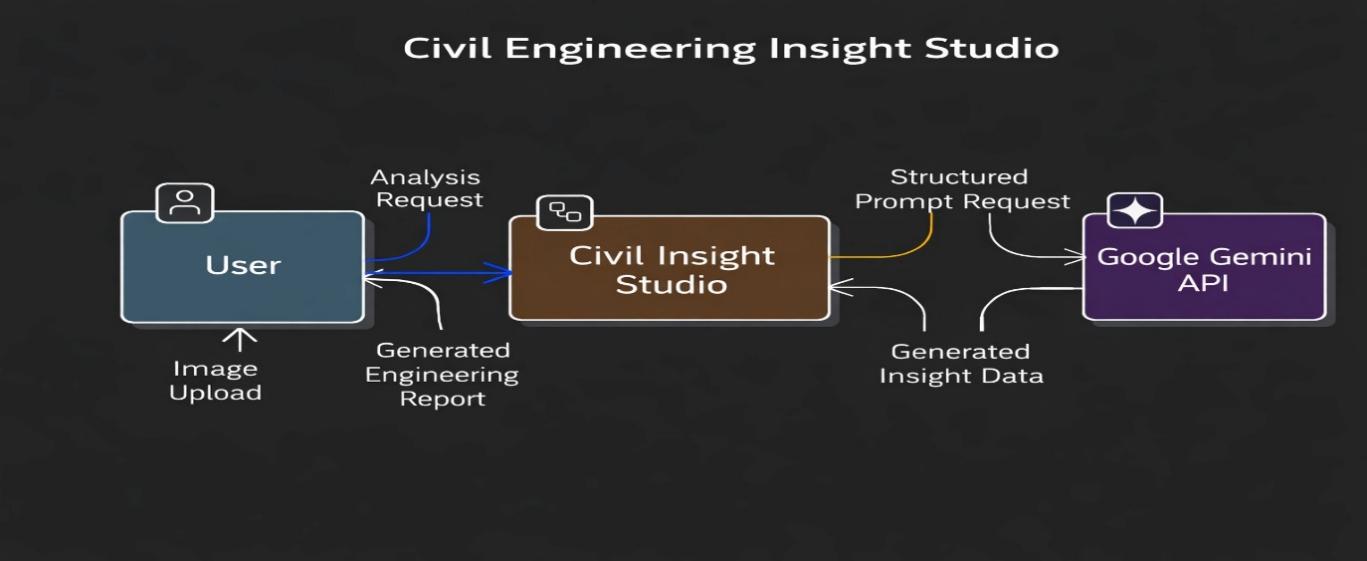
### 3.2 Non-Functional Requirements

* Usability – Simple Streamlit interface
* Performance – Response within 3–10 seconds
* Reliability – No crash during API failure
* Security – API key stored securely using environment variables
* Scalability – Cloud deployable stateless architecture

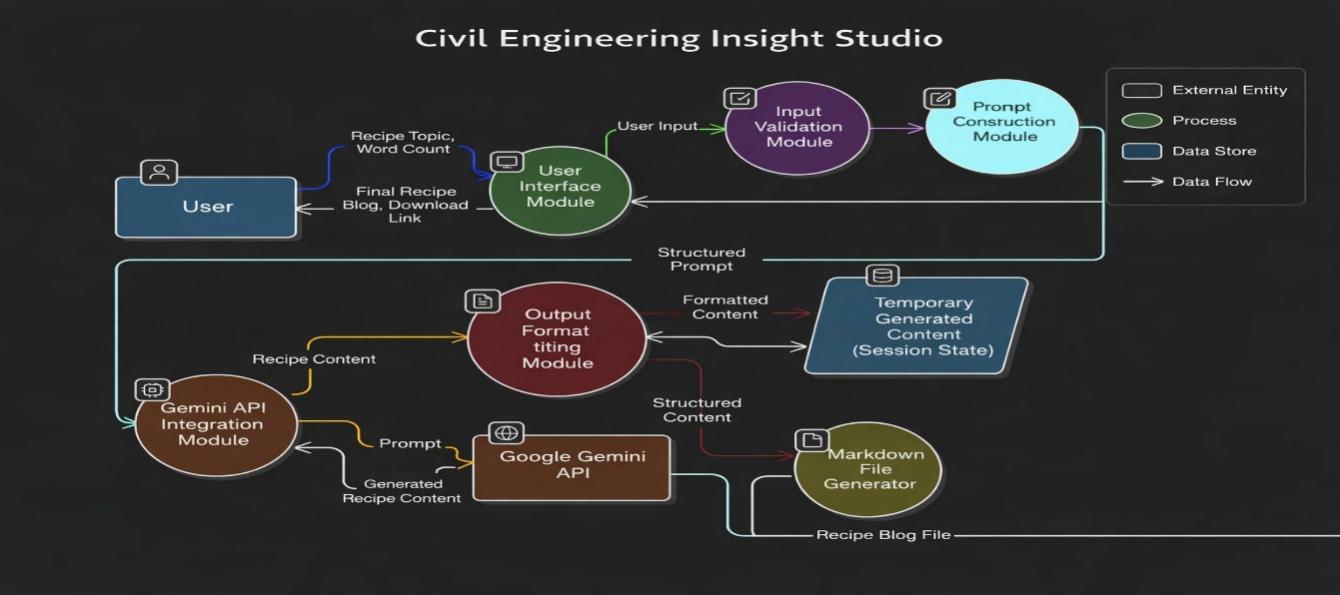
### 3.3 Technology Stack

Frontend: Streamlit  
Backend: Python  
AI Model: Google Gemini Vision API  
Deployment: Local / Streamlit Cloud

DFD Level 0 (Industry Standard)



DFD Level 1 (Industry Standard)



The system follows a stateless architecture with no persistent database.

**4. PROJECT DESIGN**

**4.1 Problem–Solution Fit**

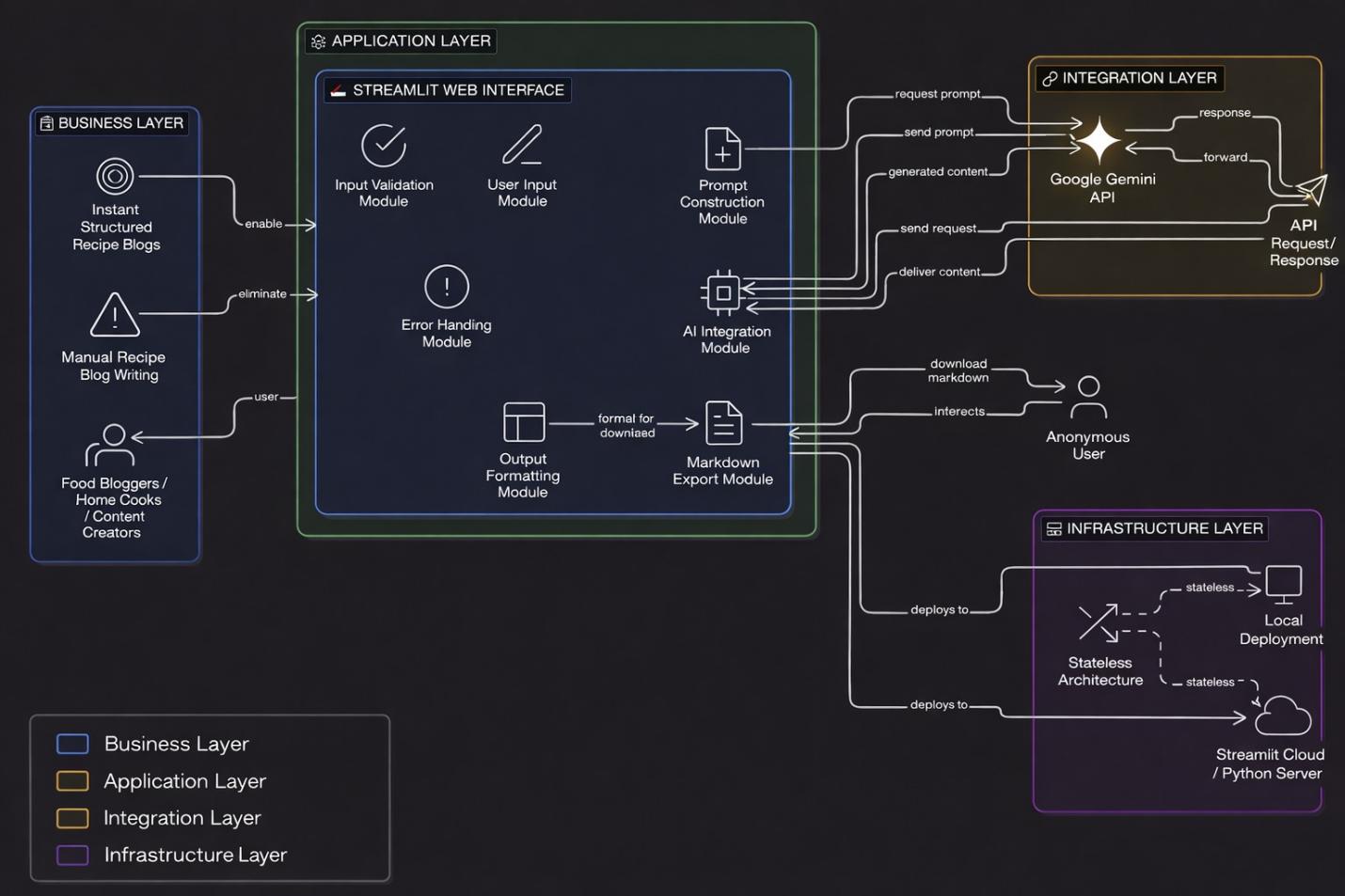
Problem: Manual structural documentation is slow and inconsistent.  
  
Solution: AI-powered image analysis generating structured engineering reports instantly.

Calendar

Description automatically generated

**4.2 Solution Architecture**

The system follows a layered architecture:  
• Client Layer – Civil Engineer (Web User)  
• Presentation Layer – Streamlit Interface  
• Application Layer – Python Logic + Prompt Engineering  
• Integration Layer – Google Gemini Vision API  
• Infrastructure Layer – Local/Cloud Hosting



**5. PROJECT PLANNING & SCHEDULING**

Development was completed in two sprints covering UI development, Gemini API integration, structured report formatting, validation, and testing.  
  
Average team velocity: 15–19 Story Points per sprint.

**6. FUNCTIONAL AND PERFORMANCE TESTING**

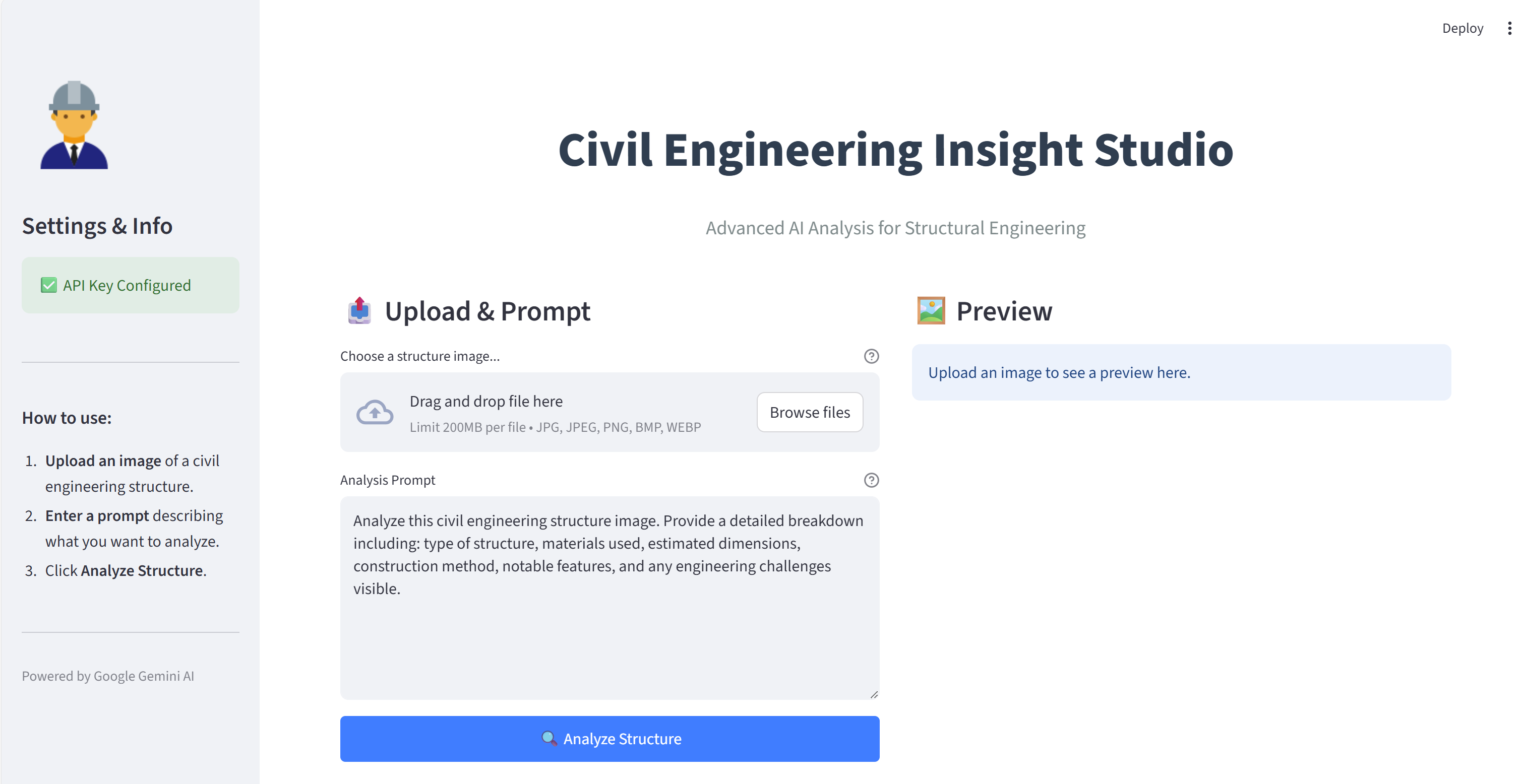
Functional tests validated image upload, API integration, and report generation.  
Performance tests confirmed average response time within 3–10 seconds depending on API latency.  
System remained stable during UAT.

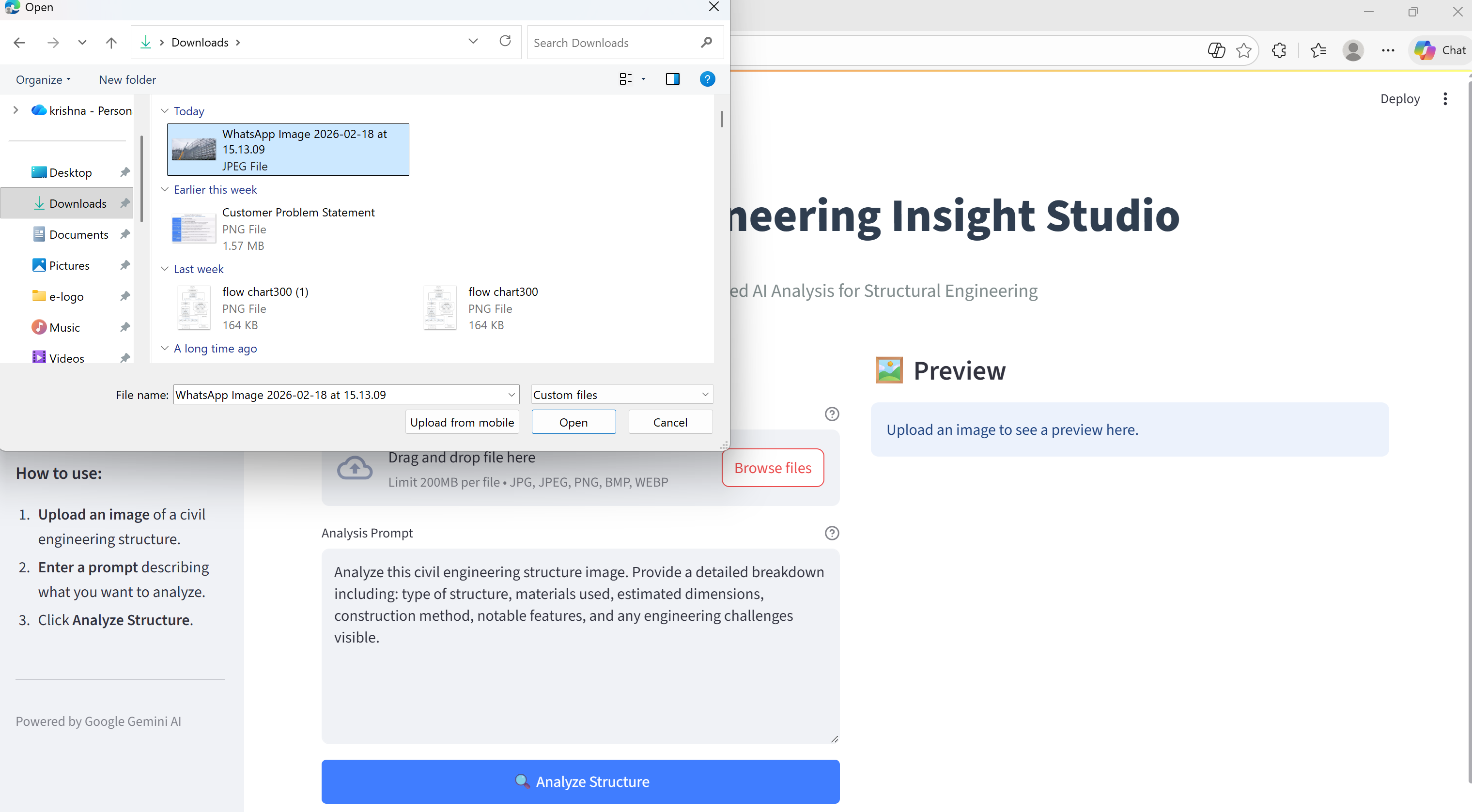
**7. RESULTS**

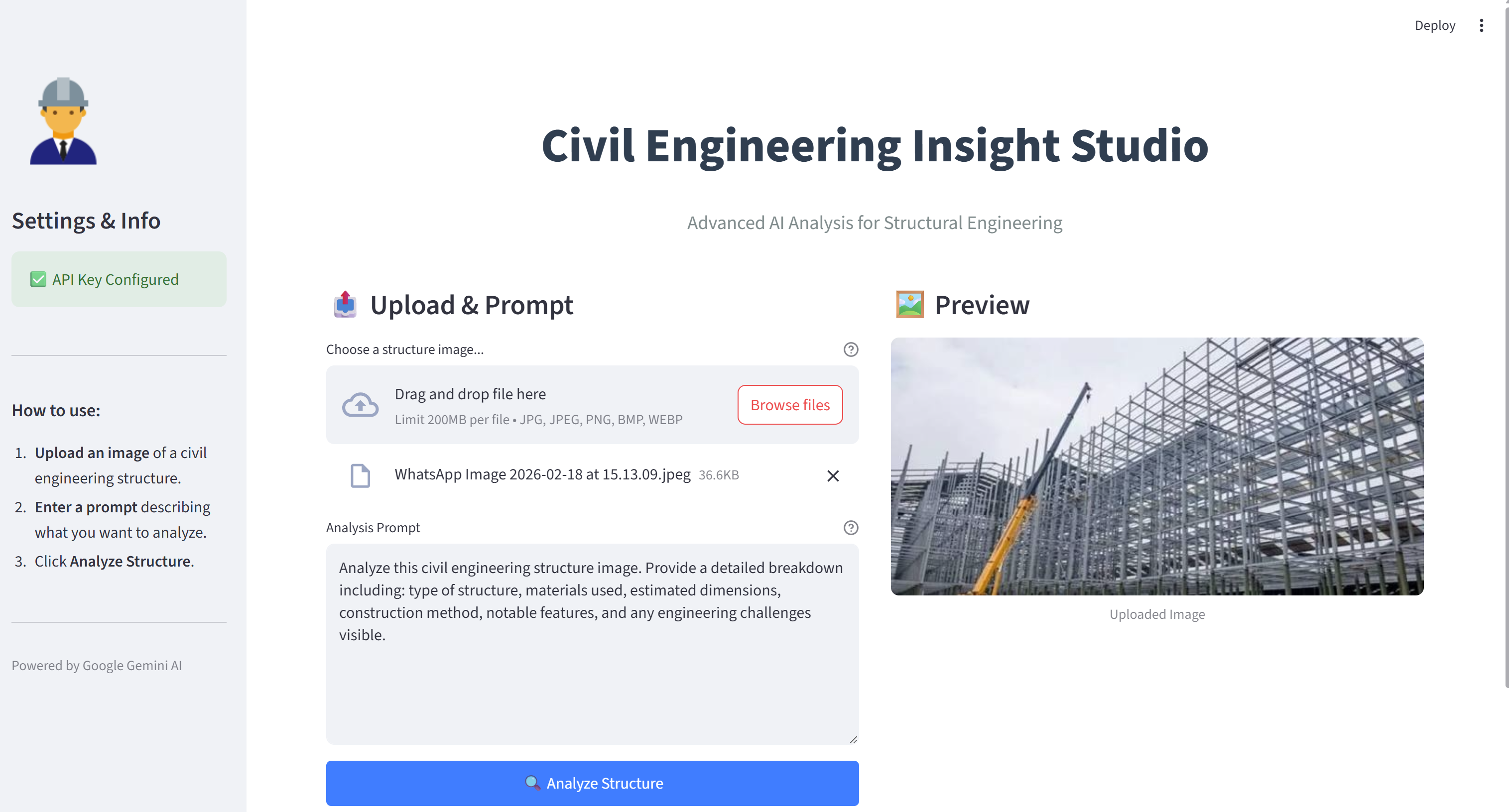
The system successfully generated structured engineering reports including:  
• Structure type identification  
• Materials detection  
• Structural components (beams, columns, slabs)  
• Construction stage observations  
  
Outputs demonstrated consistent formatting and professional documentation quality.

**7.1 Output Screenshots**

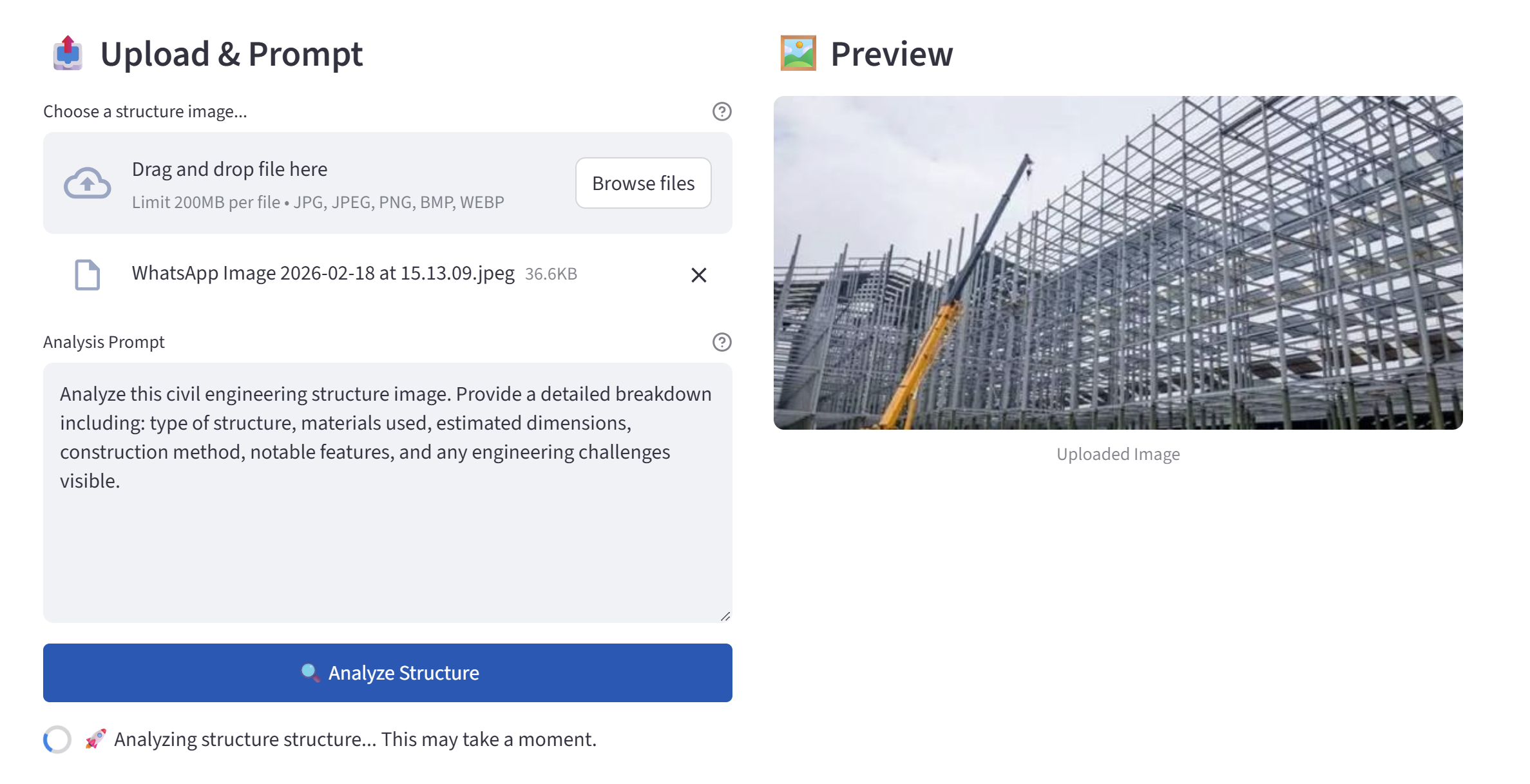
* Main interface screen



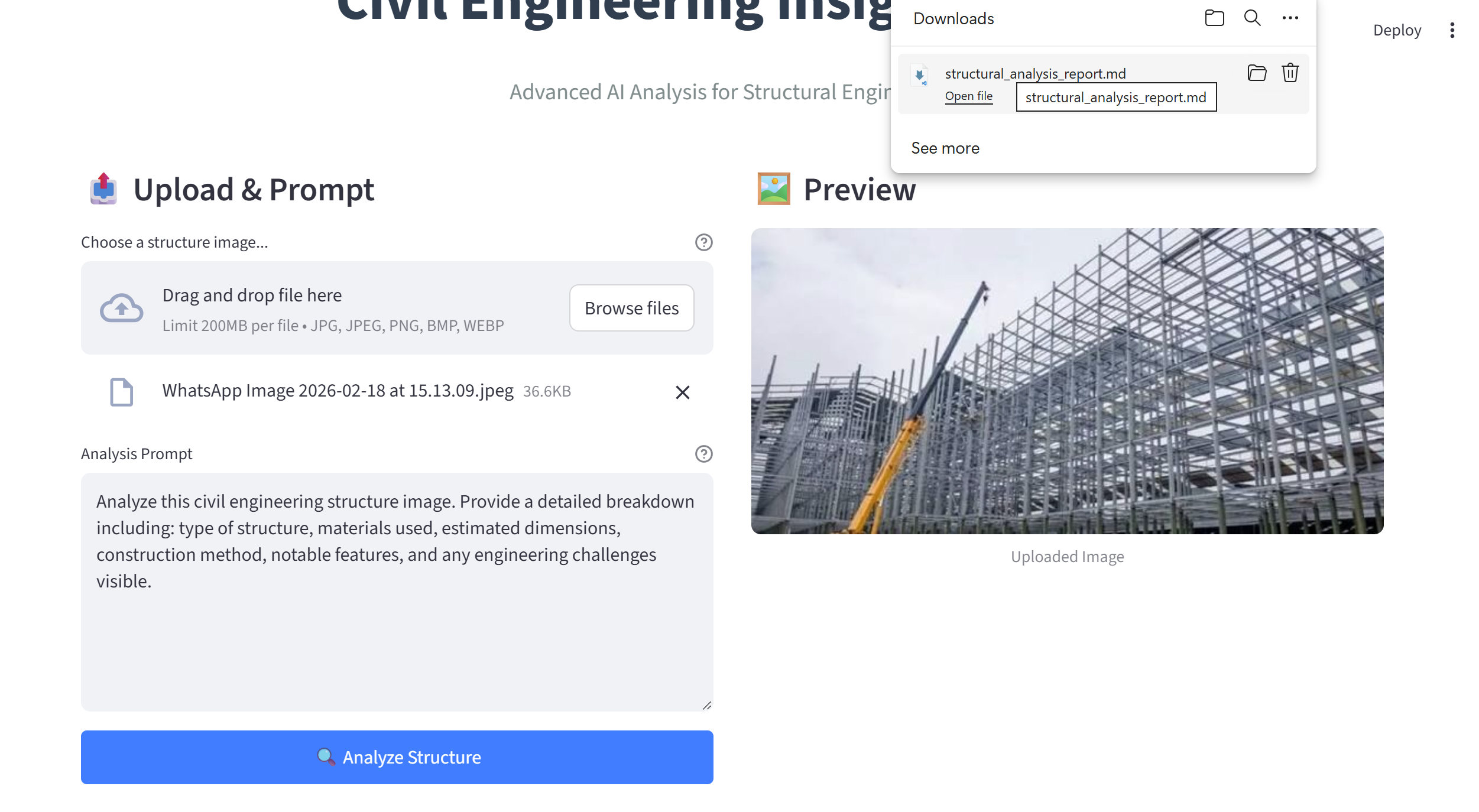
* Input section
* Upload generation



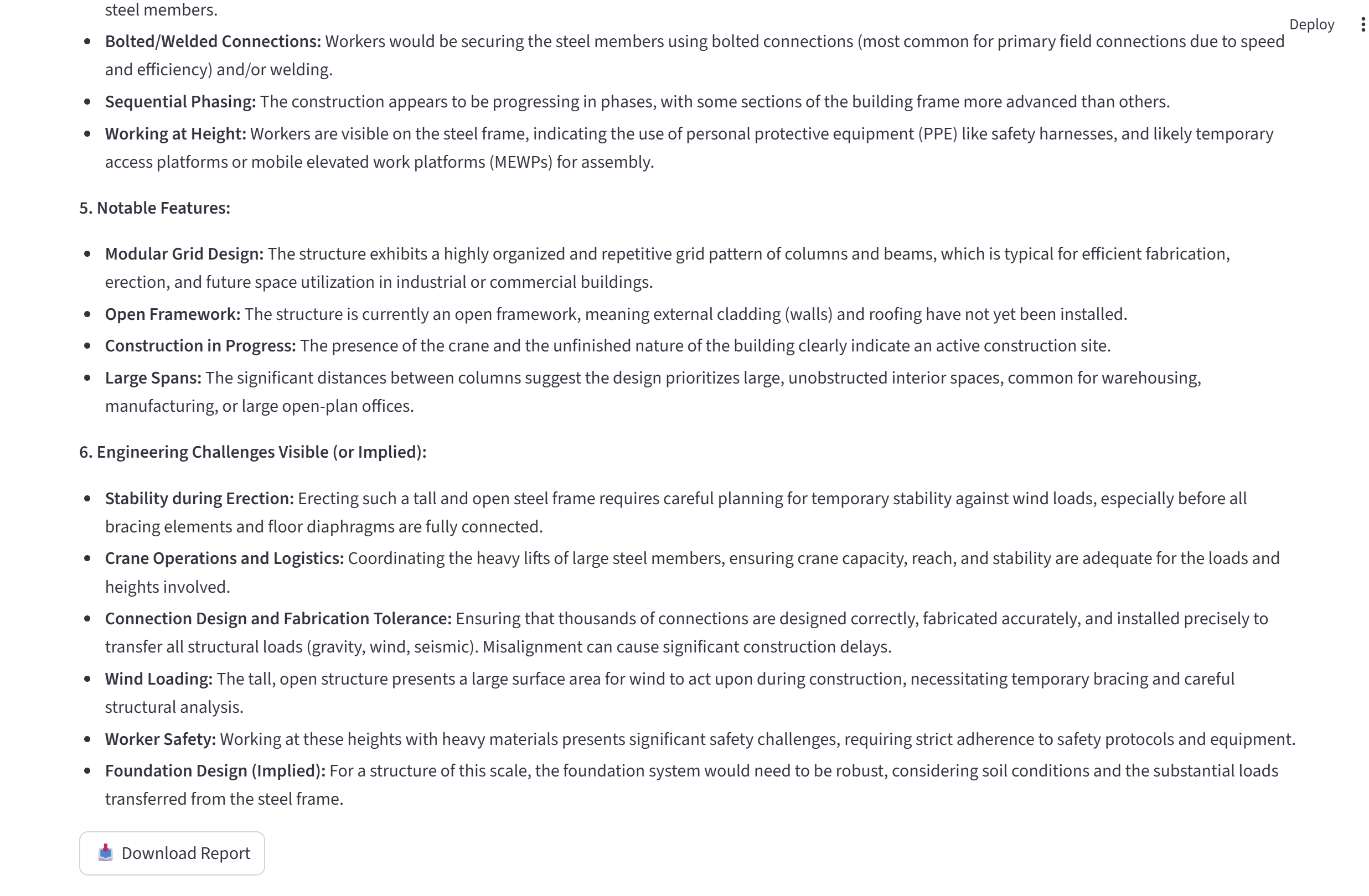
* Generated output



* Markdown download confirmation



* Output Summary



**8. ADVANTAGES & DISADVANTAGES**

**Advantages**

* Significant time savings
* Automated structured documentation
* Industry-relevant AI integration
* Cloud deployable architecture

**Disadvantages**

* Dependent on external AI API
* Requires internet connection
* API latency may vary

**9. CONCLUSION**

Civil Engineering Insight Studio demonstrates practical integration of multimodal Generative AI into civil engineering workflows. The system reduces manual documentation effort, improves reporting accuracy, and validates scalable AI-powered automation in the construction domain.

**10. FUTURE SCOPE**

* AI-based risk detection with bounding boxes
* Progress comparison between multiple images
* PDF report export feature
* User authentication & project history
* Cloud auto-scaling deployment

**11. APPENDIX**

**Source Code (app.py):**

from dotenv import load\_dotenv

import streamlit as st

import os

import google.generativeai as genai

from PIL import Image

import io

*# Load environment variables from .env file*

load\_dotenv()

*# Configure Google Generative AI with API key*

api\_key = os.getenv("GOOGLE\_API\_KEY")

genai.configure(api\_key=api\_key)

def get\_gemini\_response(input\_text, image, prompt):

*"""*

*Function to get response from Gemini model.*

*Args:*

*input\_text: User input text*

*image: List containing image data in the format required by Gemini*

*prompt: The prompt to send to the model*

*Returns:*

*str: The text response from the model*

*"""*

    model = genai.GenerativeModel('gemini-2.5-flash')

    try:

        response = model.generate\_content([input\_text, image[0], prompt])

        return response.text

    except Exception as e:

        return f"Error generating content: {e}"

def input\_image\_setup(uploaded\_file):

*"""*

*Function to read the uploaded image and format it for Gemini Pro model.*

*Args:*

*uploaded\_file: The uploaded file object from Streamlit*

*Returns:*

*list: A list containing image data formatted for Gemini API*

*Raises:*

*FileNotFoundError: If no file is uploaded*

*"""*

    if uploaded\_file is not None:

*# Read the file's binary data*

        bytes\_data = uploaded\_file.getvalue()

*# Create image parts in the required format*

        image\_parts = [

            {

                "mime\_type": uploaded\_file.type,

                "data": bytes\_data,

            }

        ]

        return image\_parts

    else:

        raise FileNotFoundError("No file uploaded")

def setup\_page():

    st.set\_page\_config(page\_title="Civil Engineering Insight Studio", layout="wide", page\_icon="🏗️")

*# Custom CSS for styling*

    st.markdown("""

        <style>

        .main {

            background-color: #f8f9fa;

        }

        .stButton>button {

            width: 100%;

            background-color: #007bff;

            color: white;

            border-radius: 5px;

            font-weight: bold;

            border: none;

            padding: 10px;

        }

        .stButton>button:hover {

            background-color: #0056b3;

            color: white;

        }

        .header-title {

            text-align: center;

            font-size: 3em;

            font-weight: 800;

            color: #2c3e50;

            margin-bottom: 0.5em;

        }

        .sub-header {

            text-align: center;

            font-size: 1.2em;

            color: #7f8c8d;

            margin-bottom: 2em;

        }

        .feature-card {

            background-color: white;

            padding: 20px;

            border-radius: 10px;

            box-shadow: 0 4px 6px rgba(0,0,0,0.1);

        }

        </style>

    """, unsafe\_allow\_html=True)

def main():

*"""Main function to run the Streamlit application."""*

    setup\_page()

*# Sidebar for configuration and info*

    with st.sidebar:

        st.image("https://img.icons8.com/color/96/000000/engineer.png", width=100)

        st.title("Settings & Info")

        if not api\_key:

            st.error("⚠️ API Key Missing")

            st.info("Please add GOOGLE\_API\_KEY to your .env file.")

        else:

            st.success("✅ API Key Configured")

        st.markdown("---")

        st.subheader("How to use:")

        st.markdown("""

        1. \*\*Upload an image\*\* of a civil engineering structure.

        2. \*\*Enter a prompt\*\* describing what you want to analyze.

        3. Click \*\*Analyze Structure\*\*.

        """)

        st.markdown("---")

        st.caption("Powered by Google Gemini AI")

*# Main Content Area*

    st.markdown('<div class="header-title">Civil Engineering Insight Studio</div>', unsafe\_allow\_html=True)

    st.markdown('<div class="sub-header">Advanced AI Analysis for Structural Engineering</div>', unsafe\_allow\_html=True)

    col1, col2 = st.columns([1, 1], gap="medium")

    with col1:

        st.markdown("### 📤 Upload & Prompt")

        uploaded\_file = st.file\_uploader(

            "Choose a structure image...",

            type=["jpg", "jpeg", "png", "bmp", "webp"],

            help="Supported formats: JPG, PNG, WEBP"

        )

        input\_prompt = st.text\_area(

            "Analysis Prompt",

            value="Analyze this civil engineering structure image. Provide a detailed breakdown including: type of structure, materials used, estimated dimensions, construction method, notable features, and any engineering challenges visible.",

            height=200,

            help="This prompt guides the AI analysis."

        )

        analyze\_button = st.button("🔍 Analyze Structure", type="primary")

    with col2:

        st.markdown("### 🖼️ Preview")

        if uploaded\_file is not None:

            image = Image.open(uploaded\_file)

            st.image(image, caption="Uploaded Image", use\_container\_width=True)

        else:

            st.info("Upload an image to see a preview here.")

*# Analysis Results Section (Full Width below columns)*

    if analyze\_button and uploaded\_file:

         if not api\_key:

             st.error("Please configure your API key first.")

         elif input\_prompt.strip() == "":

             st.warning("Please enter a prompt.")

         else:

             with st.spinner("🚀 Analyzing structure structure... This may take a moment."):

                 try:

                    image\_data = input\_image\_setup(uploaded\_file)

                    response = get\_gemini\_response(input\_text="Analyze the following image:", image=image\_data, prompt=input\_prompt)

                    st.markdown("---")

                    st.subheader("📊 Analysis Report")

                    st.markdown(response)

*# Download button*

                    st.download\_button(

                        label="📥 Download Report",

                        data=response,

                        file\_name="structural\_analysis\_report.md",

                        mime="text/markdown"

                    )

                 except FileNotFoundError as e:

                     st.error(f"File Error: {e}")

                 except Exception as e:

                     st.error(f"Analysis Failed: {str(e)}")

if \_\_name\_\_ == "\_\_main\_\_":

    main()

**GitHub Repository**

<https://github.com/nandinioduri78/Civil-Engineering-Insight-Studio>

**Project Demo Link**

[**https://drive.google.com/file/d/1CvoY4tUqxvjuFWgRchyF8DZicNLf71v8/view?usp=sharing**](https://drive.google.com/file/d/1CvoY4tUqxvjuFWgRchyF8DZicNLf71v8/view?usp=sharing)