AI PPT CONTENT GENERATOR

Team Name	Sentrio
Team ID	P-2023-27-EI-006

Team Leader Name	NEHA SHRI V	711723106070
Team Member 1	SRI JANANI S	711723106105
Team Member 2	VIGNESH BALA R	711723106121

KiTE Department Name	ECE
SOI Vertical Name	EMBEDDED & IOT

INTRODUCTION

The AI PPT Content Generator is an intelligent automation system designed to create structured PowerPoint presentations from a user-provided topic. The project is implemented using Python in Google Colab and integrates Generative Artificial Intelligence to automate research, content organization, and slide generation.

The system combines web search, web scraping, prompt engineering, and Large Language Model (LLM) capabilities to generate a complete presentation automatically. The final output is a properly formatted PowerPoint (.pptx) file containing structured slides such as title slide, overview slide, detailed content slides, and conclusion slide.

This project demonstrates a practical real-world application of Generative AI in automating academic and professional tasks. It reduces the time required to prepare presentations and improves efficiency by minimizing manual effort.

The system uses the **Google Gemini API** for intelligent content generation and dynamically retrieves updated information from online sources to improve content quality and relevance.

PROBLEM STATEMENT

Preparing a PowerPoint presentation manually requires several sequential steps:

- Searching for relevant and updated information
- Reading and understanding content
- Summarizing key points
- Structuring slides logically
- Formatting titles and bullet points
- Designing presentation layout

This process is time-consuming, repetitive, and often inefficient.

Although some AI-based tools can generate text, they usually:

- Provide unstructured output
- Do not integrate live web search
- Do not generate slides automatically
- Require manual formatting and copy-pasting

There is a need for an automated system that can:

- Perform research
- Extract relevant information

- Generate structured slide content
- Automatically create a PowerPoint file

This project solves that problem by building a complete AI-powered presentation generation pipeline.

OBJECTIVES

The primary objectives of this project are:

- Automate PowerPoint content generation.
- Use Generative AI to generate structured and logical slides.
- Integrate real-time web search for updated information.
- Reduce manual effort and preparation time.
- Generate presentation output in JSON format for structured parsing.
- Automatically convert generated content into a PowerPoint file.
- Ensure consistent formatting across slides.
- Demonstrate practical integration of AI APIs in Python applications.

GENERATIVE AI OVERVIEW

Generative Artificial Intelligence refers to AI systems that can create new content such as:

- Text
- Images
- Code
- Audio
- Video

In this project, Generative AI is used for **text generation**. A Large Language Model (LLM) is used to generate structured presentation content based on:

- User input topic
- Retrieved web content
- Carefully designed prompt instructions

The model understands the topic, analyzes contextual information, and produces organized slide content including titles and descriptive bullet points.

The project uses Google's Gemini model for text generation through API-based cloud execution.

GENAI CONCEPTS APPLIED

The following Generative AI concepts are implemented:

1. Large Language Models (LLMs)

A pre-trained large language model is used to generate high-quality structured content.

2. Prompt Engineering

Carefully designed prompts guide the model to produce:

- Exact number of slides
- Detailed bullet points
- JSON-only output

3. Structured Text Generation

The AI generates content in a predefined JSON structure to ensure reliable parsing.

4. Retrieval-Based Context (RAG-lite)

Web search results are retrieved and passed as context to improve accuracy and factual correctness.

5. API-based Model Integration

The system communicates with the Gemini model through API requests.

PROMPT ENGINEERING

Prompt engineering plays a crucial role in this project.

The structured prompt instructs the model to:

- Generate exactly 7 slides.
- Include a title slide.
- Include an overview slide.
- Provide four detailed concept slides.
- Include a conclusion slide.
- Use deep, descriptive bullet points.
- Return output strictly in JSON format.
- Avoid markdown formatting.

This ensures:

- Consistency in output structure.
- Easier JSON parsing.
- Reduced formatting errors.
- High-quality structured content.

Strict formatting instructions reduce hallucination and improve reliability.

TEXT / IMAGE / CODE GENERATION

Text Generation

This project primarily uses text generation. The Gemini model generates:

- Slide titles
- Bullet points
- Logical content flow
- Descriptive explanations

IMAGE GENERATION

No image generation is currently implemented. The system focuses entirely on structured text output.

Code Generation & Execution

Python code is used to:

- Parse JSON output.
- Create PowerPoint slides.
- Insert titles and bullet points.
- Save presentation file automatically.

EMBEDDINGS / RETRIEVAL / FINE-TUNING

This project does not use:

- Embeddings
- Vector databases
- Fine-tuning

However, it uses a simple retrieval mechanism:

- DuckDuckGo web search
- Web scraping
- Context extraction

Retrieved content is cleaned and injected into the prompt to improve accuracy and depth.

LIBRARIES & FRAMEWORKS USED

Programming Language

- Python

Platform

- Google Colab

Python Libraries Used

- google-genai – Connects to Gemini API
- python-pptx – Generates PowerPoint files
- duckduckgo-search – Performs web search
- beautifulsoup4 – Extracts webpage content
- requests – Fetches web data
- json – Parses AI output
- os – File management

GENAI FRAMEWORKS USED

- Google Generative AI SDK
- Gemini API

No Hugging Face or LangChain frameworks were used.

BACKEND / FRONTEND FRAMEWORKS

Backend:

- Python

Platform:

- Google Colab

Frontend:

- No separate frontend interface implemented.

MODELS & API USAGE

Model Used

- gemini-3-flash-preview

Model Type

- Pre-trained Large Language Model

Execution Mode

- API-based cloud execution

No local model (like Ollama) is used.

PRE-TRAINED MODELS

This project utilizes a pre-trained Large Language Model (LLM) rather than training a model from scratch. Pre-trained models are trained on massive datasets containing diverse textual information such as books, research papers, websites, and structured documents. These models learn language patterns, grammar, contextual reasoning, and knowledge representation during training.

Model Used:

- **gemini-3-flash-preview**

The model is accessed through the **Google Gemini API**.

Why Pre-trained Model?

Using a pre-trained model offers several advantages:

- No need for large-scale training infrastructure.
- Saves computational cost.
- Provides strong reasoning capability.
- Supports structured output generation.
- Enables faster development and deployment.

The pre-trained Gemini model already understands:

- Slide-based content formatting
- Hierarchical structuring
- Topic summarization
- Logical content organization

This allows the system to generate well-structured presentation slides without additional model training.

HUGGING FACE MODELS

No Hugging Face models were used in this project.

Although Hugging Face provides many open-source LLMs such as GPT-style models and transformer-based architectures, this project directly uses the Gemini API for simplicity, speed, and structured JSON output capability.

Why Hugging Face Was Not Used:

- No need for local model hosting.
- Avoid high GPU memory requirements.
- Gemini API provides structured JSON output more reliably.
- Reduced setup complexity in Google Colab.

Future versions may explore Hugging Face transformer models for offline execution.

LLM APIs Used

The system integrates the **Google Gemini API** for text generation.

API Role in the Project:

- Receives structured prompt.
- Processes contextual web content.
- Generates slide structure in JSON format.
- Returns structured response.

API WORKFLOW:

1. User topic is collected.
2. Web content is retrieved.
3. Structured prompt is prepared.
4. API request is sent.
5. JSON response is received.
6. Output is parsed.

Using API-based execution ensures:

- Fast response time.
- No need for local model installation.
- Scalable architecture.
- Cloud-based processing power.

LOCAL LLM EXECUTION

No local LLM execution is implemented in this project.

Models such as:

- Ollama
- LLaMA
- Mistral
- GPT-J

were not used locally.

Reason:

- Local LLMs require high computational resources.
- GPU memory constraints in Colab.
- Complex setup and configuration.
- Slower inference compared to optimized cloud APIs.

Cloud-based API execution was preferred for:

- Stability
- Performance
- Ease of integration

MODEL / VERSION DETAILS

Model Name:

- gemini-3-flash-preview

Model Type:

- Pre-trained Large Language Model

Provider:

- **Google**

Execution Mode:

- Cloud-based API execution

Capabilities:

- Text generation
- Structured output generation
- Reasoning
- Contextual understanding
- JSON formatting

The model supports structured prompting, which is critical for generating slide-based outputs.

MODEL SELECTION JUSTIFICATION

Gemini 3 Flash Preview was selected based on the following factors:

1. Speed

Provides low-latency responses suitable for real-time generation.

2. Structured Output

Supports reliable JSON output, which is essential for automated slide parsing.

3. Cost Efficiency

Optimized for cost-performance balance compared to larger models.

4. Reasoning Ability

Capable of generating logically structured presentations.

5. API Stability

Well-documented and stable API support.

TRADE-OFFS CONSIDERED

Several trade-offs were evaluated during model selection:

Accuracy vs Speed

Gemini Flash provides slightly lower depth than larger models but offers faster response time.

Cost vs Performance

Cloud API incurs cost, but avoids hardware investment.

Privacy vs Accessibility

Cloud-based execution requires internet and API calls.

Scalability vs Control

API-based models scale easily but provide limited customization compared to fine-tuned models.

DATASET / INPUT DESCRIPTION

This project does not use a fixed dataset.

Input:

- User-provided topic (Example: Global Warming, Artificial Intelligence, Cyber Security)

Dynamic Data Collection:

- Web search results
- Public online content

The system dynamically gathers contextual information instead of relying on pre-collected datasets.

DATASET SOURCE

The data source includes:

- DuckDuckGo web search results
- Public websites
- Educational blogs
- Informational articles

The web search provides updated and relevant information.

PREPROCESSING STEPS

To ensure high-quality AI input, the following preprocessing steps are performed:

1. Perform web search.
2. Retrieve top relevant URLs.
3. Send HTTP request to websites.
4. Extract paragraph text using BeautifulSoup.
5. Remove HTML tags.
6. Remove short or irrelevant paragraphs.

7. Limit content length.
8. Clean special characters.
9. Combine text into structured context.

This preprocessing ensures:

- Noise reduction
- Relevant content selection
- Efficient prompt size management
- Improved AI response accuracy

SYSTEM ARCHITECTURE (BLOCK DIAGRAM EXPLANATION)

The system architecture follows a pipeline-based design:

User Input
→ Web Search
→ Web Scraping
→ Context Cleaning
→ Prompt Construction
→ Gemini API Call
→ JSON Output
→ JSON Parsing
→ Slide Creation
→ PPT File Generation

Each module is independent but connected sequentially.

This modular design allows future upgrades such as:

- Adding image generation
- Adding database storage
- Adding frontend interface

PROCEDURE / METHODOLOGY

The methodology follows structured automation:

Step 1: User Input

User enters presentation topic.

Step 2: Web Search

DuckDuckGo search retrieves relevant URLs.

Step 3: Web Scraping

BeautifulSoup extracts textual content.

Step 4: Content Cleaning

Irrelevant data is removed.

Step 5: Prompt Engineering

Structured prompt is prepared.

Step 6: AI Content Generation

Gemini API generates JSON slide structure.

Step 7: JSON Validation

Response is checked for formatting errors.

Step 8: PowerPoint Creation

python-pptx creates slides.

Step 9: File Saving

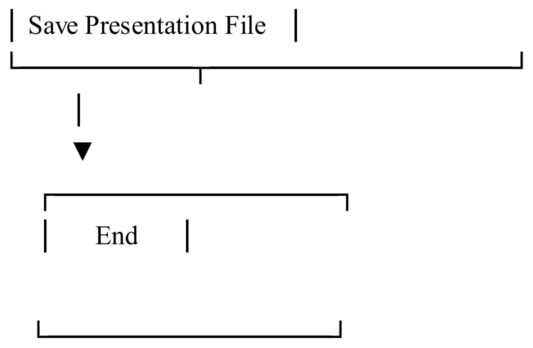
Presentation file is saved locally.

STEP-BY-STEP IMPLEMENTATION FLOW

- Install required libraries in Google Colab.
- Configure Gemini API key.
- Import required modules.
- Collect user input topic.
- Retrieve web search results.
- Extract webpage content.
- Clean and preprocess text.
- Generate structured prompt.
- Call Gemini API.
- Parse JSON output.
- Create slides using python-pptx.
- Save generated PowerPoint file.

FLOW CHART

```
┌─────────────────────────────┐
    │   Start    │
    └──────────────┬──────────────┘
         │
         ▼
┌───────────────────────────────────────┐
│ Enter Topic (User)  │
└─────────────────┬─────────────────────┘
         │
         ▼
┌──────────────────────────────────────────────────┐
│ Perform Web Search         │
│ (DuckDuckGo Search)        │
└─────────────────┬────────────────────────────────┘
        │
        ▼
┌──────────────────────────────────────────────────┐
│ Extract Web Content        │
│ (BeautifulSoup + Request)│
└─────────────────┬────────────────────────────────┘
        │
        ▼
┌──────────────────────────────────────────────────┐
│ Generate Content using     │
│ Gemini API             │
└─────────────────┬────────────────────────────────┘
        │
        ▼
┌──────────────────────────────────────────────────┐
│ Parse JSON Output          │
└─────────────────┬────────────────────────────────┘
        │
        ▼
┌──────────────────────────────────────────────────┐
│ Create PPT using         │
│ python-pptx            │
└─────────────────┬────────────────────────────────┘
        │
        ▼
┌──────────────────────────────────────────────────┐
```



IMPLEMENTATION DETAILS

Main Functions:

get_web_search_results()

- Retrieves search URLs.

extract_web_content()

- Extracts webpage paragraphs.

generate_presentation_content()

- Sends prompt to Gemini API.
- Returns structured JSON output.

create_pptx()

- Generates PowerPoint slides.
- Adds titles and bullet points.

main()

- Controls entire workflow.

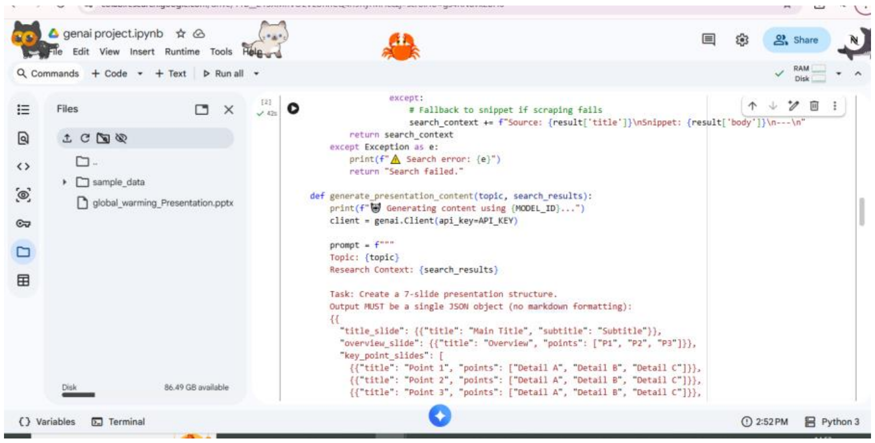
Error handling ensures robustness and prevents application crashes.

API USAGE SCREENSHOTS

The system uses API calls with:

- API key authentication
- Prompt request
- Response handling

- JSON parsing


genai project.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
Share
Files
sample_data
global_warming_Presentation.pptx
Disk 86.49 GB available
Variables Terminal
2:52 PM Python 3

```
            except:
                # Fallback to snippet if scraping fails
                search_context += f"Source: {result['title']}\nSnippet: {result['body']}\n---\n"
        return search_context
    except Exception as e:
        print(f"⚠ Search error: {e}")
        return "Search failed."

def generate_presentation_content(topic, search_results):
    print(f"🐱 Generating content using {MODEL_ID}...")
    client = genai.Client(api_key=API_KEY)

    prompt = f"""
    Topic: {topic}
    Research Context: {search_results}

    Task: Create a 7-slide presentation structure.
    Output MUST be a single JSON object (no markdown formatting):
    {{
      "title_slide": {{"title": "Main Title", "subtitle": "Subtitle"}},
      "overview_slide": {{"title": "Overview", "points": ["P1", "P2", "P3"]}},
      "key_point_slides": [
        {{"title": "Point 1", "points": ["Detail A", "Detail B", "Detail C"]}},
        {{"title": "Point 2", "points": ["Detail A", "Detail B", "Detail C"]}},
        {{"title": "Point 3", "points": ["Detail A", "Detail B", "Detail C"]}},
```

RESULTS & OUTPUT ANALYSIS

For example input: “Global Warming”

Output includes:

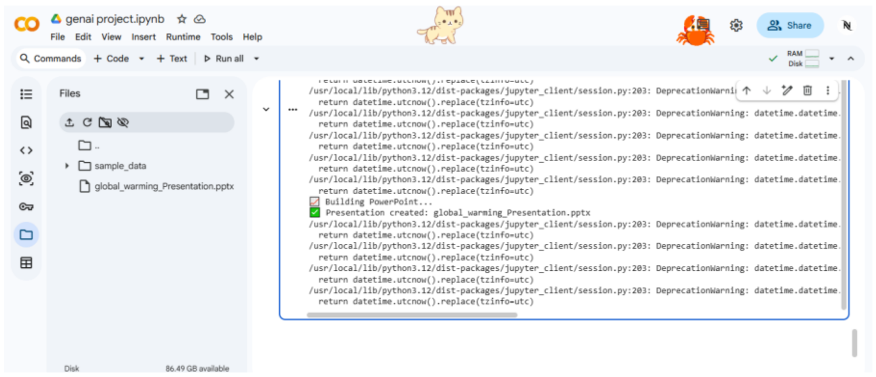
- Title Slide
- Introduction Slide
- Causes Slide
- Effects Slide
- Impacts Slide
- Mitigation Strategies Slide
- Conclusion Slide

The slides contain:

- Structured bullet points
- Logical flow
- Clear topic explanation

The generated presentation meets academic and professional standards.

SCREENSHOTS / VISUALIZATIONS



```
/usr/local/lib/python3.12/dist-packages/jupyter_client/session.py:203: DeprecationWarni
  return datetime.utcnow().replace(tzinfo=utc)
/usr/local/lib/python3.12/dist-packages/jupyter_client/session.py:203: DeprecationWarning: datetime.datetime.
  return datetime.utcnow().replace(tzinfo=utc)
/usr/local/lib/python3.12/dist-packages/jupyter_client/session.py:203: DeprecationWarning: datetime.datetime.
  return datetime.utcnow().replace(tzinfo=utc)
/usr/local/lib/python3.12/dist-packages/jupyter_client/session.py:203: DeprecationWarning: datetime.datetime.
  return datetime.utcnow().replace(tzinfo=utc)
/usr/local/lib/python3.12/dist-packages/jupyter_client/session.py:203: DeprecationWarning: datetime.datetime.
  return datetime.utcnow().replace(tzinfo=utc)
📈 Building PowerPoint...
✅ Presentation created: global_warming_Presentation.pptx
/usr/local/lib/python3.12/dist-packages/jupyter_client/session.py:203: DeprecationWarning: datetime.datetime.
  return datetime.utcnow().replace(tzinfo=utc)
/usr/local/lib/python3.12/dist-packages/jupyter_client/session.py:203: DeprecationWarning: datetime.datetime.
  return datetime.utcnow().replace(tzinfo=utc)
/usr/local/lib/python3.12/dist-packages/jupyter_client/session.py:203: DeprecationWarning: datetime.datetime.
  return datetime.utcnow().replace(tzinfo=utc)
/usr/local/lib/python3.12/dist-packages/jupyter_client/session.py:203: DeprecationWarning: datetime.datetime.
  return datetime.utcnow().replace(tzinfo=utc)
```

EVALUATION / PERFORMANCE ANALYSIS

Performance metrics:

- Fast generation (few seconds)
- High structured consistency
- Accurate topic coverage
- Minimal formatting errors

Limitations:

- Internet dependency
- API rate limits
- No graphical customization
- No citation referencing yet

CHALLENGES FACED

1. JSON formatting inconsistencies
2. Markdown removal issues
3. Web scraping timeouts
4. API rate limit errors
5. Handling empty search results

Solutions implemented:

- Strict prompt constraints
- Response cleaning
- Exception handling
- Fallback logic

CONCLUSION

The AI PPT Content Generator successfully automates presentation creation using Generative AI. The integration of web search, AI text generation, and PowerPoint automation creates a powerful end-to-end solution.

The system reduces manual effort, improves productivity, and demonstrates practical AI application in real-world tasks.

FUTURE ENHANCEMENTS / SCOPE

- Add automatic image generation
- Add PPT design themes
- Add slide transition effects
- Build web-based application
- Add user login system
- Add multilingual support
- Add citation referencing
- Add voice-to-topic input
- Implement embedding-based retrieval
- Deploy as SaaS platform

REFERENCES

- Gemini API Documentation
- Python-PPTX Documentation
- DuckDuckGo Search Library
- BeautifulSoup Documentation

GITHUB LINK

Repository:
https://github.com/NEHASHRIV26/Gen-Ai_Sentrio

ADDITIONAL OUTPUTS

The project produces the following outputs:

- Generated PowerPoint (.pptx) file
- Google Colab notebook
- Structured JSON output
- Prompt template
- Execution screenshots

- API response logs
- System architecture diagram
- Flowchart representation