Project Report: An Smart Building Design Slide Generator Based on RAG

Yiyang Pan
University of Wisconsin Madison
pan248@wisc.edu

Abstract

This paper provides a general overview of designing a simple slide generator for the Smart Building Design industry using language models.

1 Introduction

With the rapid growth in the field of natural language processing, many everyday tasks can be automated using a combination of tools such as language models and plugins. This project aims to meet the need for automating the daily procedures of Smart Building Systems Designers by creating systems that generate slides according to clients' needs.

2 Problem Statement

Creating design plans for different projects is one of the most time-consuming tasks for smart building systems designers. Typically, the design consists of a 100-page slide deck with detailed descriptions, including background information, needs analysis, system design plans, required equipment, and past experiences on similar projects. Even though specific project details, such as the area covered, may differ, many plans share a significant amount of similar information. As we develop an online database to help new designers find past projects for reference, we aim to take it a step further by automating the draft generation process. This would provide designers with a nearly complete plan, thus streamlining their workflow.

3 Aims and Objectives

The goal of this project is to develop a system that can automatically generate new slides based on user input. The new design should be professional and adhere to the industry standards of smart building systems design; therefore, the entire slide deck should follow a certain format and logical flow. The design plan should also be consistent with the user's input and the content of past documents, with each slide having a clear central topic. Finally, given that many of the required documents are under copyright protection, the process must be able to run locally within a reasonable time frame.

4 Solution Approach

4.1 Overview

We designed a slide generation tool demo. First, we preprocess the document into markdown format, then segment the sentences according to length, embed the text into vectors using embedding models, and store them in a vector database. After this, the server is set up. Upon user input, the frontend calls the server's API to generate results.

4.2 Pre-processing and File Loader

Since most past documents are in PPT format, we initially considered using the python-pptx tool to load them directly without preprocessing. However, we discovered that this method often combined loosely related text from multiple pages, which is not ideal. By preprocessing the slides into markdown, we can add breaks for each page, include page numbers, and detect page titles. The preprocessed markdown files are then used for segmentation. Additionally, the preprocessing process is synchronized with the original file, meaning any additions, edits, or deletions in the original file are reflected in the markdown file.

4.3 Text Segmentation and Embedding

After starting the server, we load the preprocessed markdown files along with other types of files using a universal loader. When reading from markdown, we separate text into chunks from each slide page to maintain the original boundaries. We chose the lightweight all-MiniLM-L6-v2 model from the GPT4ALL library for simplicity. After embedding,

the data is stored in the Chroma database locally, prepared for later RAG processing(a20, 2024).

4.4 Model Choice for Content Generation

We examined the behavior of three small language models: Llama 3: 8b from Meta, Phi-3: 3b from Microsoft, and Qwen2: 7b from Alibaba Group. All three models demonstrated good semantic ability, but since our project primarily deals with the Chinese language, we chose Qwen2 for its superior Chinese language capabilities.

4.5 Template and Chain Design

We designed six different chains for various purposes:

- The outline chain for designing slide outlines based on user input prompts, assisted by generated background information and RAG. After the user modifies and confirms the outline content, each item in the outline is passed to the next chain to generate content based on previous documents, ensuring professionalism and reducing hallucination.
- A chain to generate titles for the entire presentation and each slide. The title for the overall presentation is passed into the content chain along with the outline segment to ensure it remains on topic, especially for generating content like the project background and previous examples. Titles for each slide are generated after the content is finalized to provide a good overall summary.
- The fifth chain formats the output according to the design, but since it needs to output the entire slide content with minor changes, it is time-consuming and wasteful of computational resources. Therefore, we use a regular formatting template instead.
- The last chain summarizes the page content into keywords, which are then used to look up pictures using the Unsplash API. This makes pages with less content more visually appealing.

4.6 Frontend, Slide Backend, and APIs

For testing purposes, we built a simple web server including a single HTML page for API testing, as well as outline parsing and editing features. For this demo, we chose to use Slidev(Fu, 2020), an

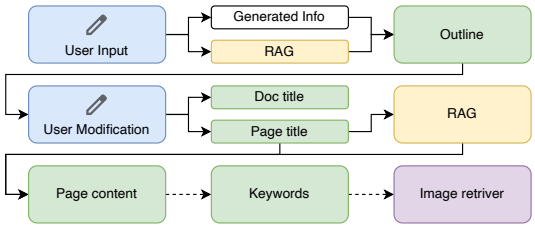

Figure 1: Chain design for the slide generator

open-source presentation slide maker, as it uses simple markdown grammar to format output with elegant templates and strong built-in tool support based on Node.js. This is intuitive for developers and can later be implemented to support various charts, animations, and embedded tools using only a few lines of code, which language models excel at. If needed, we can later replace it with slide modification tool APIs.

5 Results

Overall, the project results were ideal. The slide maker is able to generate the outline locally within one minute on an Apple M1 chip with 16GB of unified memory. The content is generally relevant, including background information, plans, previous similar projects, and conclusions. Each page has its central topic, and the generator can produce related content without going off-topic. However, sometimes the generated content may be duplicated or overlap across different slides. We can address this by adding memory capabilities to the application in the future.

6 Discussion and Analysis

During this project, we specifically analyzed the best way to use language models for slide generation. Unlike essays and other documents, which are composed paragraph by paragraph with continuous semantic meaning, the content in slides is more isolated. Instead of organizing words into lengthy paragraphs, slides are more likely to be composed of short paragraphs, bullet points, charts, graphs, and pictures.

6.1 Limitations of Smaller Language Models

Current models prefer long, continuous paragraphs. For example, when generating an outline, the model tends to generate an introductory paragraph and a concluding paragraph along with the required

bullet points, necessitating post-processing to extract the correct parts.

For content generation, the model often produces essay-like paragraphs, which are challenging to present on a single slide and not preferred by viewers. We addressed this issue using prompt engineering techniques to specify the length of the result. Using phrases like "give 1-3 sentences" proved more effective than "give no more than 3 sentences," as the latter seemed to have no effect. Our hypothesis is that smaller models struggle with vague terms such as "about," "no more than," and "at most."

Describing output format is also a challenge for prompt engineering. Smaller models, like Qwen2, often struggle to adhere to specified formats. For example, when prompted to "use markdown format... do not include titles," the output still frequently contains titles. This issue may relate to prompt engineering techniques, which suggest avoiding negative instructions and instead directly telling the model what to do.

We attempted to solve this problem by providing examples, but another issue arose: the model sometimes treated the example content as the "content" retrieved by the RAG step or "context." After several attempts, we ended up not specifying examples and formatting requirements. This way, the model mostly outputs plain text and markdown-style lists. We then use code to inject formatting phrases into the output, providing a more reliable result.

Another problem when designing prompts for smaller models is deciding whether to separate the process into smaller steps or build a more complex prompt to achieve the goal at once. For more complex prompts, one problem is consistency. We discovered that when using complex prompts with smaller models, sometimes the model can achieve the goal, but often it just goes off-topic. However, when dividing the process into smaller steps, sometimes it just repeats the previous result with minor modifications. If the result is large, this will be a very time-consuming process.

After experiencing various combinations, we discovered that it is better for smaller models to generate a small step at a time. To speed up the process, we let models output only a short paragraph in the process before the final generation and move the task of formatting to traditional pattern matching algorithms. This keeps the process fast with more stable output formats shaped by each step in the chain.

6.2 Different Methods of Text Segmentation

Given the different form factors of slides, the retrieval process is crucial. Most of the time, text extracted directly from slides does not make much sense without the original format. We addressed this problem by using attributes such as text size and properties to tag text as "title." Also, content on one slide is supposed to be related even if it does not share similar semantic meaning. We compared using regular text splitters and semantic splitters, finding that the regular splitter outperformed the semantic splitter. To further improve retrieval results, we added separators during the preprocessing of PPT files. When splitting, we only segment the content of each slide. In many cases, the chunk size is large enough to contain all the information from the slide, allowing the retriever to access all related text content from that slide.

7 Conclusion and Future Work

The final version of the demo can provide detailed, professional outlines and related content at a decent speed. This system can significantly aid smart building system designers in enhancing their work efficiency. Designers can then edit based on the generated content.

There is still much that can be improved in the future. For one, when building the RAG component, we can further explore the file loader to improve the quality and relevance of the input data. We can also experiment with different file loaders. There is much more to discover in languages other than English for these two parts. Additionally, to improve the quality of the generated content, we can add memory to the process to make the content more consistent. Lastly, we can experiment with different models (especially larger models) and fine-tune them for better results.

References

2024. rag-chroma-private | langchain.

Anthony Fu. 2020. Slidev.