Automated PPT Generation with MCP

Author: Roshan Kumar

Objective

Design and prototype an AI-powered solution that ingests business documents **RFPs, technical proposals, and brand-identity assets(styling guidelines)** and outputs a polished, McKinsey-style slide deck suitable for executive presentations.

Introduction & Solution Overview

I have developed an automated system for generating slide decks from reports and proposals using Generative AI (GenAI), while preserving structural consistency, brand tone, and analytical rigor.

The solution ingests content from various sources - including PDFs and web pages then performs the following steps:

1. **Content Extraction & Thematic Analysis:** The system parses the input data to extract text, then uses a ranking algorithm to identify and prioritize key themes based on frequency and relevance. The top *k* themes (default: 5) are selected for deeper analysis.
2. **Insight Extraction via LLMs:** For each identified theme, I used an LLM (leveraging a local database via the Model Context Protocol, MCP) to extract key insights and summarizations, ensuring domain relevance and contextual accuracy.
3. **Presentation Generation Pipeline**: A secondary MCP server handles the slide generation process, which includes:
    a. A slide layout engine that selects optimal slide types (e.g., title, content-heavy, chart, comparison) based on the content.
    b. Dynamic content positioning to maximize clarity and visual appeal.
    c. Application of a Brand Guidelines System to enforce color schemes, fonts, styles, brand voice compliance, and confidentiality level tagging.
4. **Content Governance & Quality Control**:
    To ensure reliability and professionalism, I implemented:
    a. Confidentiality screening
    b. Hallucination risk assessment
    c. Brand voice validation
    d. Content quality scoring

This end-to-end automation significantly reduces manual effort, improves data accuracy, and ensures all output maintains a consistent, polished, and consultative tone aligned with brand standards.

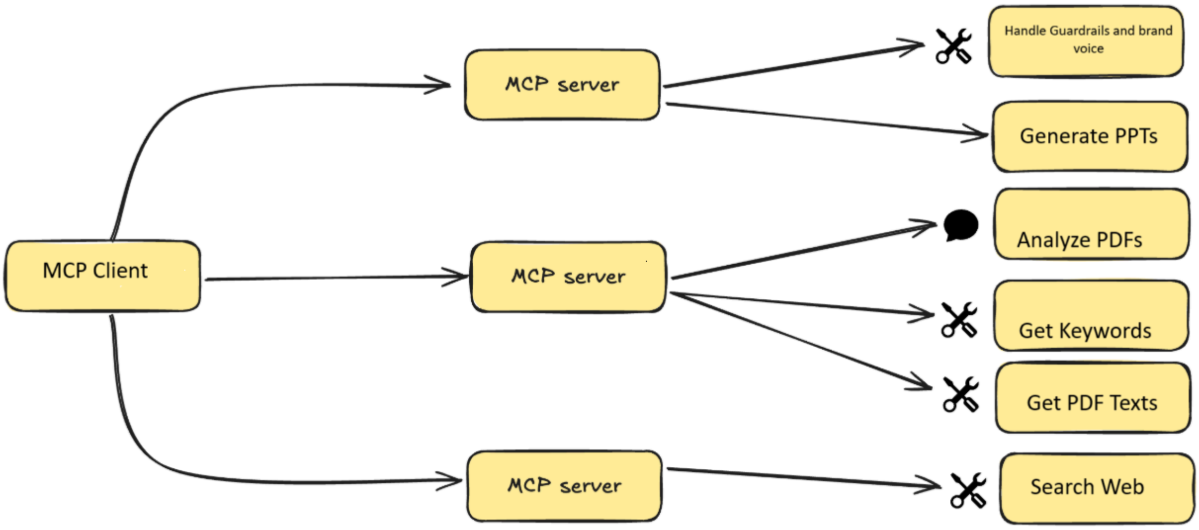
System Architecture

We adopt a microservices approach using MCP (Model Context Protocol) servers. MCP follows a client-server model where a host (e.g. Claude Desktop) can connect to multiple specialized servers .

Our system comprises three main MCP servers: -

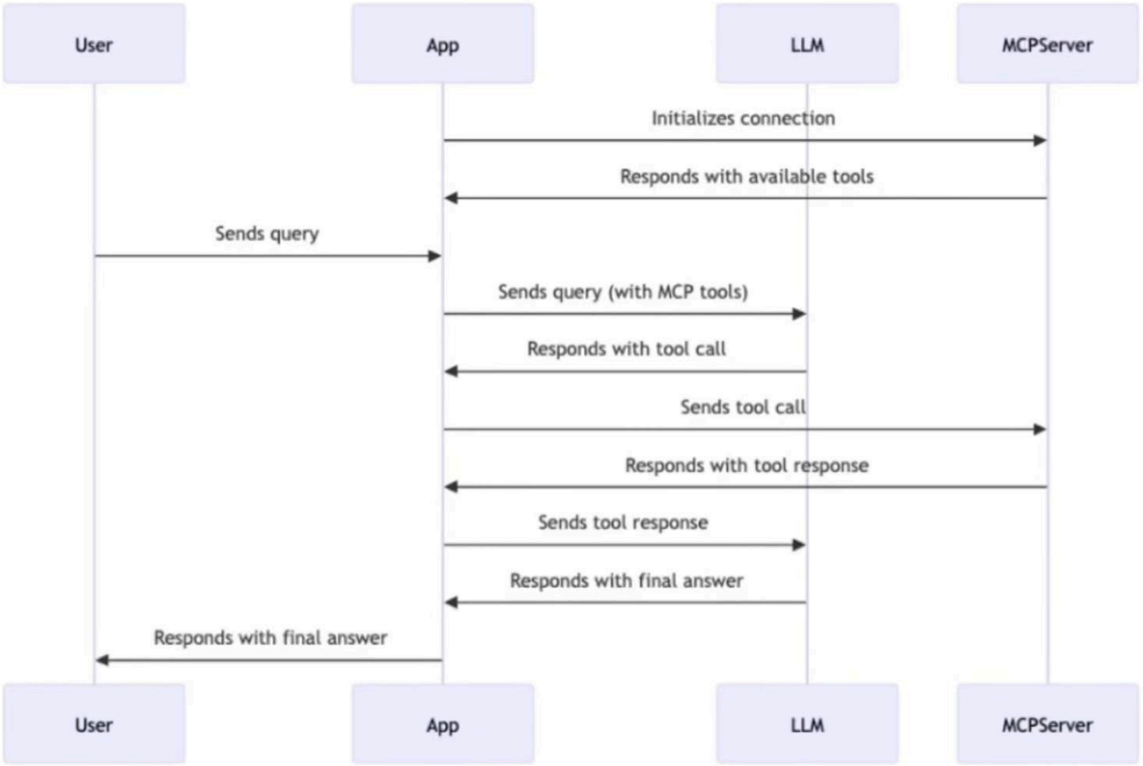
1. **pdf_analyzer**: Extracts text and tables from PDF files. It uses a Python library PyMuPDF and PyPDF2 for parsing, organizing content by page/section. It serves three tools to user : `analyze_pdf, extract_pdf_text, get_pdf_keywords`
2. **documentation**: Ingests content from URLs. It uses a **SERPER API** to fetch realtime Google search results and page HTML. The API runs in a headless browser and returns JSON with page text and metadata . We then extract the main article text for analysis.
3. **ppt_generator**: It takes preprocessed content chunks and generates slide structures (titles, bullets, charts) via LLM prompts, enforcing layout and style rules. Each server is a lightweight MCP application providing a specific capability .

The Claude Desktop host (MCP client) connects to all servers, sending document context and queries. Below is a conceptual multiserver architecture diagram (placeholder): Figure: Example microservice architecture pattern.



Each service handles a separate function (PDF parsing, web ingestion, slide generation).

- MCP Host/Client: Claude Desktop or similar AI interface. It holds API keys (e.g. SlideSpeak API) and triggers servers via STDIO.
- Data Sources: Local files (for PDF/DOCX) and remote APIs (SERPER API) feed into the respective servers.
- Communication: MCP transports are with **STDIO Transport**. Each server returns structured data (e.g. parsed text, summary) for the LLM to consume.


User
App
LLM
MCPServer
Initializes connection
Responds with available tools
Sends query
Sends query (with MCP tools)
Responds with tool call
Sends tool call
Responds with tool response
Sends tool response
Responds with final answer
Responds with final answer
User
App
LLM
MCPServer

Technical Approach

Document Ingestion Pipeline

1. **Supported Formats:** We handle PDF and DOCX inputs. For PDF, we use PyMuPDF and PyPDF2 to extract text per page. Each document is first converted into plain text with paragraph structure.
2. **NLP Preprocessing**: We tokenize and clean the text, removing stopwords and stemming where needed. We then compute TF-IDF vectors for each paragraph/chunk using a tool like `sklearn.feature_extraction.text, TfidfVectorizer`. TF-IDF reweights term frequencies by inverse document frequency, emphasizing unique terms in context . We apply **KMeans clustering** on these TF-IDF vectors to identify thematic sections (e.g., topics, proposal sections). This groups related paragraphs so that each cluster represents a potential slide content area.
3. **Chunking Logic**: To feed content into LLMs, we split the document into manageable “chunks” of text. Following best practices, each chunk is sized so it “makes sense without surrounding context” (roughly 500–1000 tokens, depending on model). This avoids dropping semantic meaning. We ensure each chunk ends at logical boundaries (paragraph or sentence). We allowed slight overlaps or sliding windows to cover context edges.
4. **Specialized Extraction:** The PDF parser tags figures and tables, capturing captions and numeric data. The link parser server first obtains web page HTML via SERPER Api, then uses `BeautifulSoup4` to scrape the page and get article text. SERPER

APIs can retrieve the “knowledge graph” or main content quickly (in ~1-2s) , so pages are ingested in near real-time. If clustering is incomplete, we fall back to simpler TF-IDF to pick top-relevance paragraphs.

LLM Orchestration & RAG Logic

1. **Prompt Flow & Summarization:** We implement a multi-stage prompt chain. First, a summarizer prompt ingests each chunk (or cluster) and produces a bullet-point summary of that section. Then a higher-level prompt sequences these summaries into an outline for slides (title + key points per slide). Finally, a slide-generation prompt takes each outline item and generates final slide content (bullets, chart descriptions, captions).
2. **Retrieval & Filtering**: For RAG (Retrieval-Augmented Generation), we vectorize all chunks and store them in an index. When generating a particular slide, we retrieve the top-k most relevant chunks (e.g. by cosine similarity) to use as context. We also apply focus-area filtering: if the input specifies a topic (e.g. “marketing strategy”), we bias retrieval toward chunks tagged with that theme. If key information is missing, a fallback is to broaden the search (increase k or switch clustering). Each retrieved chunk is verified against query keywords to avoid irrelevant content.
3. **Hallucination Control**: We integrate safety prompts and verification steps. For example, after generating slide text, the LLM is prompted to cite supporting sections (e.g. “Which pages of the source contain this information?”). This echo-check prevents fabricated facts. Research shows that RAG can still hallucinate if it retrieves noisy docs , so we enforce strict filtering and “Chain-ofVerification” in prompts. If a generation step seems to rely on no source, we ask the model to mark it as uncertain or drop it. We also constrain the model to speak in confidence only when evidence exists.

Slide Layout Engine

1. **Analyze Slide Template:** Analyze an uploaded slide template image to extract design elements and structure.
2. **Validate presentation content**: Validate presentation content for brand voice compliance, confidentiality, and quality.
3. **Page Hierarchy & Templates**: Slides follow a structured flow (title, context, insights, next steps). We use predetermined layout templates for each type. For example, a “Three-Bullets” template has one title box and three bullet boxes, while a “Chart” template reserves space for a figure and caption. These templates come from brand design guidelines (standard fonts, colors, logo placement). The system chooses the template based on slide class.

Brand Voice & Confidentiality Safeguards

1. **Voice Style Enforcement**: The LLM is instructed to write in a consistent brand voice (e.g. “professional and friendly tone” for client-facing decks). We can embed brand guidelines into prompts (colors, phrasing, allowed jargon). Tools like create_brand_guidelines can programmatically enforce style by rejecting or

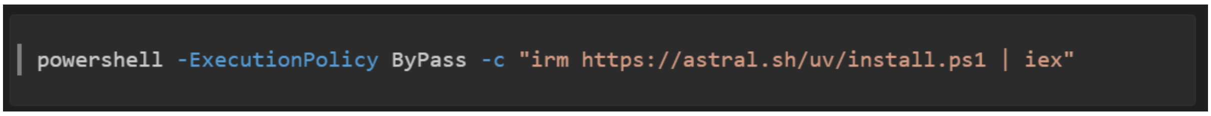
rephrasing output that deviates from a style checklist. Some products even fine-tune a “brand LLM” to internal docs , which could further lock in tone.

2. **Confidentiality Tier Detection**: Before generation, the document is auto-classified for sensitivity. We use AI-based data classification (not just regex) to detect if content is “public,” “internal,” or “confidential.” Modern systems can scan text and metadata to label sensitive data without needing manual rules . If a document is labeled confidential, the slide generator omits sharing of explicit data (only high-level insights), or encrypts parts of the slide. This ensures no sensitive numbers leak into a deck unless appropriate.
3. **Brand Compliance Validators**: After slides are generated, we run compliance checks: correct logo usage, approved color palette, font sizes, etc. A small script compares slides to company-approved slide masters. Any violations (e.g. off-brand fonts) trigger a reformat step. This retains a “McKinseystyle” consistency (clean layout, corporate colors).

Prototype Implementation

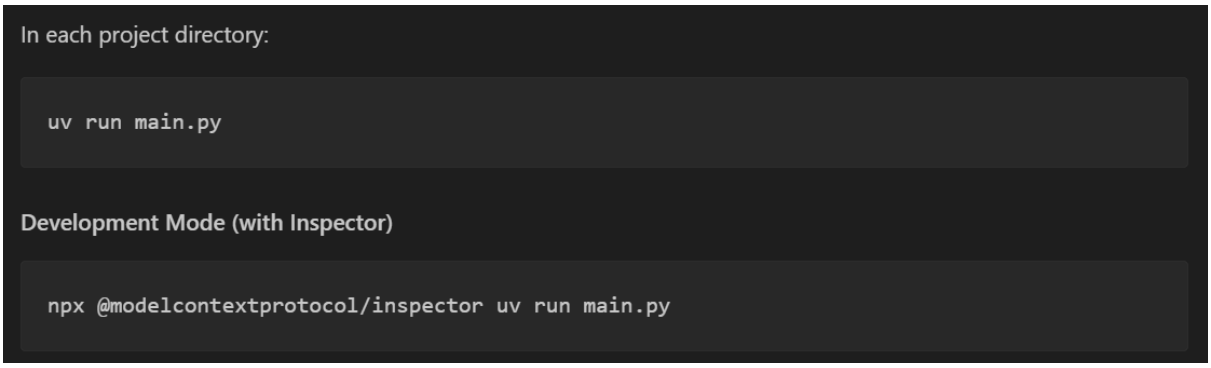
We built a prototype using Claude Desktop as the MCP host and a suite of MCP servers. Our SlideSpeak MCP servers handle presentation tasks.

1.Install uv



```
powershell -ExecutionPolicy ByPass -c "irm https://astral.sh/uv/install.ps1 | iex"
```

2. Install and Setup MCP Servers
3. Install Dependencies using “uv”
4. Configure Claude Desktop
5. Run the servers



In each project directory:

```
uv run main.py
```

Development Mode (with Inspector)

```
npx @modelcontextprotocol/inspector uv run main.py
```

Limitations and Next Steps

My prototype is a proof of concept, but several gaps remain.

#TODOs

1. Integrate server with custom MCP client.
2. Automate the steps to and stitch the server responses together.
3. Build an UI on streamlit with capabilities like file upload, chat feature and polished prompts.
4. Build a more robust slide generator engine

Improvements: Currently, the link parser uses generic TF-IDF rather than a true semantic embedding for clustering. We plan to implement embeddings (via OpenAI or Cohere) for better focus selection. Integrating the backends into a cohesive interface is in progress; we aim to build a web dashboard (e.g. with Streamlit) where users upload files and tweak slide parameters. The slide-generation ML logic is rule-based for now; we foresee using feedback loops (allowing users to upvote slide drafts) to refine prompts and templates. Performance can improve by parallelizing chunk analysis and caching frequent queries. Finally, real McKinsey decks often use polished diagrams; automating chart formatting and adding diagram generation (perhaps via DALL·E or mermaid.js) is on our roadmap.

Suggestions for Improvement

- **Automated Diagram Generation**: We could integrate an AI diagram tool to convert bulleted data into charts or flow diagrams (e.g. using a graph-plotting library or AI chart generation), further reducing manual edits.
- **Style Transfer Learning**: A future enhancement is fine-tuning an LLM on approved brand content, creating a “brand LLM” that inherently writes in company style . This would automate compliance even more deeply than prompt instructions.
- **Content Scoring & Feedback**: Implement an automated content-quality score (based on coherence, factuality, brand fit) and allow users to provide feedback. The system could then adjust slide generation parameters dynamically, effectively “learning” from usage.

By anchoring generation in retrieved content and enforcing corporate style, we ensure decks are both accurate and on-brand. This demonstration highlights key GenAI product engineering skills in LLM orchestration, microservice design, and prompt engineering. Sources: We leveraged the Model Context Protocol (MCP) framework , SERPER APIs , and best practices in document chunking and retrieval-augmented generation . Industry reports highlight manual slide creation as a major bottleneck , motivating this automated approach. MCP tools and Claude Desktop enabled easy integration . Automated classification literature informed our confidentiality safeguards . These references guided the design of our pipeline and safety measures.