Gemini

LaTeX Engines and Markdown Conversion

https://gemini.google.com/share/221ac52b0bdc

Some responses below were generated according to the creator's saved info and preferences. Learn more February 4, 2026 at 06:56 PMPublished February 4, 2026 at 07:14 PM

Opens in a new window

what are differences between pdflatex and xelatex?

How to install pdflatex?

How to install xelatex?

How are they related to pandoc?

what are the best way to convert from markdown to latex to pdf, especially,

when .md doc has images and tables

Choosing between pdflatex and xelatex often comes down to how much you enjoy wrestling with font packages. Since you are working on your paper, **"Chinese Characters as Living Fossils,"** this choice is actually critical: xelatex is almost certainly what you need for proper Unicode and CJK (Chinese, Japanese, Korean) support.

---

1. pdflatex vs. xelatex

The primary difference lies in how they handle fonts and character encoding.

Feature	pdflatex	xelatex
Font Support	Uses legacy TeX fonts (Type 1). Difficult to use system fonts (TTF/OTF).	Supports **Modern System Fonts** (TTF/OTF) natively via fontspec.
Encoding	Traditionally 8-bit. Needs packages	Native **UTF-8** support.

Feature	pdflatex	xelatex
	like inputenc for UTF-8.	
Multilingual	Requires complex setups for non-Latin scripts.	Excellent for **Chinese characters** and complex scripts.
Speed	Generally faster.	Slower due to font processing.

> **Bottom Line:** For a paper involving Chinese characters, **xelatex** is the gold standard because it allows you to use any font installed on your computer (like *SimSun* or *Kaiti*) with a single command.

---

2. How to Install Them

You don't usually install these individually. They are "engines" included in a **LaTeX Distribution**. Installing a distribution gives you both, along with `pandoc`.

- **Windows:** Install MiKTeX (lightweight) or TeX Live (complete).
- **macOS:** Install MacTeX.
- **Linux:** Use your package manager (e.g., `sudo apt install texlive-full`).

---

3. Relation to Pandoc

Pandoc acts as a "translator." It doesn't create PDFs itself; instead:

1. Pandoc converts your **Markdown** into a **LaTeX** file.
2. It then calls an underlying engine (like `pdflatex` or `xelatex`) to turn that LaTeX file into a **PDF**.

By default, Pandoc uses `pdflatex`. For your specific project, you must tell Pandoc to use `xelatex` to ensure your characters render correctly.

---

4. Best Way to Convert (MD → LaTeX → PDF)

When dealing with images and tables, the "best" way is to use a **Metadata Block (YAML)** and a specific command-line string.

The Recommended Workflow

1. **Prepare your Markdown:** Include a YAML header at the very top of your `.md` file to handle the Chinese fonts.

   YAML

   ```yaml
   ---
   title: "Chinese Characters as Living Fossils"
   author: Your Name
   mainfont: "SimSun" # Or any Chinese font on your system
   CJKmainfont: "SimSun"
   geometry: margin=1in
   ---
   ```

2. **The Command:** Run this in your terminal:

   Bash

   ```bash
   pandoc input.md -o output.pdf --pdf-engine=xelatex
   ```

Handling Images and Tables

- **Images:** Use standard Markdown syntax: `![Caption](path/to/image.png)`. Pandoc automatically handles the LaTeX `\includegraphics` logic and scaling.
- **Tables:** Use **Pipe Tables**. They are the most stable for conversion.

  Markdown

  ```markdown
  | Character | Meaning | Evolution |
  | :--- | :--- | :--- |
  | 山 | Mountain | Pictographic |
  ```

- **The Secret Weapon:** Use the `--listing` flag if you have code blocks, or `--filter pandoc-crossref` if you need to reference "Figure 1" or "Table 2" inside your text.

Would you like me to generate a sample LaTeX template that specifically optimizes the layout for your "Living Fossils" paper?

I am actually developing a documentation agent which uses agentic RAG, it generates .md file nicely, but have trouble with generating either .docx word doc or .pdf, I am using AWS sagemaker studio env

Operating within **AWS SageMaker Studio** adds a layer of complexity because the default environment lacks the heavy LaTeX binaries required for PDF generation. For a documentation agent, you need a robust, automated pipeline.

Since you're writing about **Chinese Characters**, `xelatex` is non-negotiable for the PDF side, while `.docx` conversion is handled natively by Pandoc but requires specific image-handling flags.

---

1. Setting up the Environment (SageMaker Studio)

You can't easily install a full 5GB TeX Live distribution every time a notebook starts. Instead, use a **Lifecycle Configuration (LCC)** or run these commands in a terminal/notebook cell to set up a persistent environment.

Installation Script

Run this once in your SageMaker terminal to get the necessary binaries:

Bash

```bash
# Update and install Pandoc
sudo apt-get update
sudo apt-get install -y pandoc

# Install a "lightweight" TeX distribution (TinyTeX is recommended for cloud envs)
wget -qO- "https://yihui.org/tinytex/install-bin-unix.sh" | sh
```

```
# Add to PATH (add this to your .bashrc to make it persistent)
export PATH="$HOME/bin:$PATH"

# Install the XeTeX engine and CJK packages for Chinese support
tlmgr install xetex
tlmgr install collection-langchinese
```

---

2. The "Agentic" Conversion Strategy

To handle **Images** and **Tables** without breaking, your agent should not just run a simple command. It needs to manage **Resource Paths**.

Recommended Python Integration

If your agent is written in Python, use the `subprocess` module to call Pandoc. This allows you to dynamically set the `--resource-path`, which tells Pandoc where to find the images generated by your RAG agent.

Python

```python
import subprocess
import os

def convert_document(input_md, output_file, images_dir):
    # Determine format based on extension
    fmt = "pdf" if output_file.endswith(".pdf") else "docx"
    
    cmd = [
        "pandoc", input_md,
        "-o", output_file,
        f"--resource-path={images_dir}", # CRITICAL for images
        "--standalone"
    ]
    
    if fmt == "pdf":
        cmd += [
            "--pdf-engine=xelatex",
            "-V", "mainfont=Noto Sans CJK SC", # Ensure this font is in your env
            "-V", "geometry:margin=1in"
        ]
    elif fmt == "docx":
        # Optional: Use a reference doc to keep your company branding/styles
        # cmd += ["--reference-doc=template.docx"]
        pass

    try:
        subprocess.run(cmd, check=True)
        return f"Successfully generated {output_file}"
    except subprocess.CalledProcessError as e:
        return f"Error: {e}"
```

---

3. Solving Common "Agentic" Failures

Images: The Path Problem

Agents often generate Markdown with relative paths (e.g., `![alt](image.png)`). If the conversion script runs in a different folder, it fails.

- **Fix:** Always use the `--resource-path` flag pointing to your agent's `assets/` or `images/` folder.

- **Fix:** Ensure your agent uses standard Markdown `![]()` syntax, not HTML `<img>` tags, as LaTeX engines often ignore raw HTML.

Tables: The Formatting Problem

If your tables are getting "cut off" in PDF:

- **Fix:** Use the `longtable` package in LaTeX. Pandoc does this by default, but you can force it by adding `--variable=classoption:oneside` or using a custom LaTeX template if the tables are extremely wide.

Word (.docx) Specifics

Word is much more forgiving than PDF. If images appear in Word but not PDF, the issue is your LaTeX engine. If they don't appear in either, your `resource-path` is wrong.

Pro-Tip for SageMaker

If your SageMaker environment is frequently deleted, consider building a **Custom Docker Image** that has `pandoc`, `xelatex`, and your preferred Chinese fonts pre-installed. This ensures your agent always has its "printing press" ready without a 10-minute install wait.

Would you like a sample YAML metadata block that specifically handles the table-wrapping and Chinese font settings for your PDF output?

yes

To ensure your documentation agent produces professional PDFs—especially with the linguistic complexity of **Chinese characters** and the structural headache of **wide tables**—you should have your agent prepend this YAML block to the top of the generated `.md` file.

This block tells Pandoc and `xelatex` exactly how to behave without needing complex command-line arguments.

The Metadata Block (YAML)

YAML

```
---
title: "Chinese Characters as Living Fossils"
subtitle: "Agentic RAG Documentation"
author: "Generated by Documentation Agent"
date: \today
header-includes:
  - |
    \usepackage{xeCJK}
    \setCJKmainfont{Noto Sans CJK SC} % Replace with available system font
    \usepackage{booktabs}   % For professional looking tables
    \usepackage{longtable}  % For tables that span multiple pages
    \usepackage{caption}    % Better control over image/table captions
    \captionsetup{font=small,labelfont=bf}
geometry: "margin=1in"
mainfont: "DejaVu Sans"     % Good for English text alongside CJK
fontsize: 11pt
---
```

Why this specific setup works for your project:

- **`xeCJK` & `setCJKmainfont`:** This is the "magic" for your paper. Instead of generic LaTeX font errors, this allows `xelatex` to render Chinese characters perfectly alongside Latin text.
- **`longtable` & `booktabs`:** Standard Markdown tables often break if they are too long or wide. `longtable` ensures that if your RAG agent generates a 50-row table of character evolutions, it will flow across pages rather than disappearing into the footer.
- **`geometry`:** SageMaker/Linux default LaTeX margins are often huge. This tightens them to 1 inch to make the documentation look modern.

The "Agentic" Workflow Strategy

Since you are in a SageMaker environment, the agent should follow this 3-step execution logic:

1. **Check for Font**: Ensure the font defined in `setCJKmainfont` actually exists in the environment. On Amazon Linux (SageMaker), you might need to run `fc-list :lang=zh` to see what is installed.
2. **The Pandoc Call**:

   Bash

   ```
   pandoc input.md \
     -o output.pdf \
     --pdf-engine=xelatex \
     --table-of-contents \
     --number-sections
   ```

3. **Error Handling**: If `xelatex` fails (common in cloud envs due to missing `.sty` files), have your agent fallback to `.docx` generation, which is much less "picky" about system dependencies:

   Bash

   ```
   pandoc input.md -o output.docx
   ```

One Final Tip for Images

If your agent generates images via Matplotlib or DALL-E/Nano, ensure it saves them as **PNG** or **JPG**. While LaTeX loves PDFs/EPS, Pandoc handles PNG/JPG best when converting to both Word and PDF simultaneously.

Would you like me to provide a Python "wrapper" function that checks for these LaTeX dependencies before your agent attempts the conversion?

yes
Building a robust "pre-flight check" into your agent is smart. In a cloud environment like SageMaker, the most common reason for failure is a missing binary or a missing LaTeX package (like `xeCJK`).

The following Python script acts as a diagnostic layer. It checks for `pandoc` and `xelatex`, and if they are missing, it provides the specific shell commands to fix the environment.

The Agentic Pre-Flight Script

Python

```python
import subprocess
import shutil
import os

def check_agent_dependencies():
    """Checks if the SageMaker environment is ready for MD -> PDF conversion."""
    
    dependencies = {
        "pandoc": "Pandoc (Document Converter)",
        "xelatex": "XeLaTeX (PDF Engine)",
    }
    
    missing = []
    
    print("--- Environment Check ---")
    for cmd, name in dependencies.items():
        path = shutil.which(cmd)
        if path:
            print(f"    {name} found at: {path}")
        else:
            print(f"    {name} is NOT installed.")
            missing.append(cmd)

    if not missing:
        print("    Environment is ready for PDF generation.")
        return True
    else:
        print("\n--- Fix Commands for SageMaker Terminal ---")
        if "pandoc" in missing:
            print("Run: sudo apt-get update && sudo apt-get install -y pandoc")
        if "xelatex" in missing:
            print("Run: wget -qO- 'https://yihui.org/tinytex/install-bin-unix.sh' | sh")
            print("Then add to path: export PATH=\"$HOME/bin:$PATH\"")
        return False

def check_fonts():
    """Checks for Chinese font support."""
    try:
        # Checks if any CJK fonts are registered with the system
        result = subprocess.run(['fc-list', ':lang=zh'], capture_output=True, text=True)
        if result.stdout.strip():
            print("    Chinese fonts detected.")
            # Print the first available font name for the YAML
            first_font = result.stdout.split(':')[1].split(',')[0].strip()
            print(f"Recommended font for YAML: {first_font}")
        else:
            print("⚠ No Chinese fonts found. PDF might show boxes instead of characters.")
    except FileNotFoundError:
        print("    'fontconfig' not installed. Cannot check fonts.")

# Run the check
if check_agent_dependencies():
    check_fonts()
```

Understanding the Pipeline Architecture

Since you are using Agentic RAG, your architecture likely follows this flow to ensure images and tables are preserved:

1. **RAG Output:** The agent generates a `.md` string.
2. **Asset Management:** The agent saves any generated images/plots to a local `/tmp/assets/` folder.
3. **YAML Injection:** The agent prepends the YAML block I provided earlier, ensuring the `resource-path` points to that `/tmp/assets/` folder.
4. **Conversion:** The agent calls the `subprocess` to run Pandoc.
5. **Artifact Storage:** The resulting `.pdf` or `.docx` is uploaded to S3 or provided as a download link.

Pro-Tip for Tables in SageMaker

If your RAG agent uses **Pandas** to handle data before turning it into Markdown, use `df.to_markdown()` with the `tablefmt="grid"` or `tablefmt="pipe"` option. Pandoc reads these formats most reliably when converting to LaTeX.

Would you like me to show you how to bundle these dependencies into a Dockerfile specifically for a SageMaker Custom Image so you never have to install them manually again?

yes
Creating a custom Docker image for your SageMaker Studio environment is the "pro" move. It ensures that every time your documentation agent spins up, `pandoc`, `xelatex`, and the necessary Chinese fonts are baked into the infrastructure. No more 10-minute installation scripts or broken PDF builds.

Below is a production-ready `Dockerfile` and the steps to register it in SageMaker.

---

1. The Dockerfile

This uses a Python base image, installs the system-level dependencies (Pandoc and TeX Live), and sets up the CJK fonts.

Dockerfile

```
# Use the SageMaker distribution base or a standard Python image
FROM public.ecr.aws/sagemaker/sagemaker-distribution:latest-cpu

USER root

# 1. Install Pandoc and system utilities
RUN apt-get update && apt-get install -y \
    pandoc \
    fontconfig \
    wget \
    && rm -rf /var/lib/apt/lists/*

# 2. Install TinyTeX (lightweight LaTeX) as root
# We install to a global location so all users can access it
RUN wget -qO- "https://yihui.org/tinytex/install-bin-unix.sh" | sh -s - --admin --instpath /opt/tinytex

# 3. Add TinyTeX to the PATH
ENV PATH="/opt/tinytex/bin/x86_64-linux:${PATH}"

# 4. Install specific LaTeX packages needed for Chinese and Tables
RUN tlmgr install \
    xeCJK \
    fontspec \
    booktabs \
    longtable \
    caption \
    environ \
    trimspaces \
    etoolbox

# 5. Install Google Noto Sans CJK SC (Chinese Simplified) fonts
RUN apt-get update && apt-get install -y fonts-noto-cjk && fc-cache -fv

# Switch back to the default SageMaker user
USER sagemaker-user
WORKDIR /home/sagemaker-user
```

---

2. How to Deploy to SageMaker

To use this image in SageMaker Studio, follow these steps:

1. **Build and Push to ECR:** Use the SageMaker terminal or your local CLI to build the image and push it to **Amazon Elastic Container Registry (ECR)**.
2. **Attach to Domain:** * Go to the **SageMaker Console** -> **Images**.

- Create a new image pointing to your ECR URI.
- Attach this image to your **SageMaker Domain** as a "Custom Image."

3. **Launch:** When you open a new notebook or start your agent’s app, select this custom image as the **Image** and **Kernel**.

---

3. The Resulting Workflow

Once this image is active, your agent can run the conversion with zero setup. Here is how the internal architecture looks:

Updated Python Agent Logic

With the environment guaranteed, your agent's conversion function becomes much simpler:

Python

```python
def agent_generate_pdf(md_content, output_path):
    # The YAML header is now guaranteed to work because the fonts are in the Docker image
    header = """---
mainfont: "Noto Sans CJK SC"
CJKmainfont: "Noto Sans CJK SC"
pdf-engine: xelatex
---
```

```
"""
    full_md = header + md_content
    
    with open("temp.md", "w") as f:
        f.write(full_md)
        
    # Standard command - no extra font paths needed
    os.system("pandoc temp.md -o " + output_path + " --pdf-engine=xelatex")
```

Why this is better for your "Living Fossils" paper:

- **Consistency:** The character rendering of "山" or "水" will look identical every time the agent runs.
- **Portability:** You can move this agent to AWS Lambda or an EC2 instance by just moving the Docker image.
- **Reliability:** You avoid the "works on my machine" problem common with LaTeX fonts.

Would you like me to help you write a script that automatically uploads these generated PDFs to an **S3 bucket** so your agent can provide a temporary download link to users?

yes

To complete your agent's workflow, you need a bridge between the **SageMaker ephemeral storage** and the **user**. Since files generated in Studio are local to the instance, uploading them to S3 and generating a **Pre-signed URL** is the most efficient way to share the final `.pdf` or `.docx`.

The S3 Uploader and Link Generator

This script takes the local path of the generated document, moves it to S3, and returns a secure, temporary link that lasts for 1 hour.

Python

```python
import boto3
import os
from botocore.exceptions import ClientError

def upload_and_link(local_file_path, bucket_name, expiration=3600):
    """
    Uploads the PDF/DOCX to S3 and returns a temporary download URL.
    """
    s3_client = boto3.client('s3')
    file_name = os.path.basename(local_file_path)
    
    try:
        # 1. Upload the file
        print(f"Uploading {file_name} to S3 bucket: {bucket_name}...")
        s3_client.upload_file(local_file_path, bucket_name, file_name)
        
        # 2. Generate a pre-signed URL for the user
        url = s3_client.generate_presigned_url(
            'get_object',
            Params={'Bucket': bucket_name, 'Key': file_name},
            ExpiresIn=expiration
        )
        return url
    except ClientError as e:
        print(f"S3 Error: {e}")
        return None

# Usage in your Agent
# pdf_url = upload_and_link("output.pdf", "your-sagemaker-output-bucket")
# print(f"Download your paper here: {pdf_url}")
```

The Final Documentation Pipeline

Now that we have all the pieces—the LaTeX engine, the Docker environment, the Pandoc conversion, and the S3 delivery—here is the high-level architecture of your Documentation Agent.

Summary of the Agent's "Living Fossil" Workflow:

1. **Generation:** Agent retrieves RAG context and writes `.md` content (including your Chinese character analysis).
2. **Formatting:** Agent injects the **YAML header** specifying `xeCJK` and `Noto Sans CJK SC`.
3. **Conversion:** Agent calls `pandoc` via the `xelatex` engine (baked into your Custom Docker Image).
4. **Storage:** Agent uploads the `.pdf` to S3.
5. **Delivery:** Agent provides the user with a **Pre-signed URL** to download the polished document.

A Final Hint for Tables

If your RAG agent creates tables that are still too wide for a standard A4 page, you can add this line to your YAML header:

YAML

```
header-includes:
  - \usepackage{pdflscape}
```

Then, your agent can wrap wide tables in `\begin{landscape}` and `\end{landscape}` tags directly in the Markdown to rotate just those specific pages.

Would you like me to create a "Master Script" that combines the Markdown generation, the conversion, and the S3 upload into one single class?

before you consolidate various steps, can you clarify how to create more permanent URL from S3 file

In AWS S3, a "permanent" URL is effectively a direct link to an object in a bucket that has **Public Read** permissions. Unlike pre-signed URLs, these do not expire; however, they require you to lower the security "walls" of your bucket slightly.

For your agentic RAG system, you have two main ways to achieve this.

Option 1: The "Public Bucket" Approach (Easiest)

This is best if the documents your agent generates (like your "Living Fossils" paper) are intended to be public.

1. **Disable "Block Public Access":** In the S3 Console for your bucket, go to **Permissions** and turn off "Block *all* public access."
2. **Apply a Bucket Policy:** Add this JSON to the **Bucket Policy** section (replace `your-bucket-name` with your actual bucket name):

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "PublicReadGetObject",
            "Effect": "Allow",
            "Principal": "*",
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::your-bucket-name/*"
        }
    ]
}
```

3. **The Permanent URL Format:** Once public, your URL will always follow this standard structure: `https://your-bucket-name.s3.amazonaws.com/your-file-name.pdf`

Option 2: The CloudFront Approach (Professional & Secure)

If you want a permanent URL but want to keep the S3 bucket itself private (the AWS best practice), you use **Amazon CloudFront** as a CDN front-end.

- **How it works:** CloudFront sits in front of S3. You give CloudFront permission to read from S3 (via an Origin Access Control - OAC), and the public accesses the file via the CloudFront URL.
- **The Permanent URL:** `https://d12345678.cloudfront.net/your-file-name.pdf`
- **Benefit:** It’s faster, supports custom domains (like `docs.yourpaper.com`), and keeps your S3 bucket secure from direct "guessing" attacks.

Comparison for your Agent

Feature	Pre-signed URL	Public S3 URL	CloudFront URL
Lifespan	1 min – 7 days	**Permanent**	**Permanent**
Security	High (temporary)	Low (Public)	High (OAC protected)

Feature	Pre-signed URL	Public S3 URL	CloudFront URL
Complexity	Low (Python code)	Low (Console settings)	Medium (Setup CDN)
Use Case	Private user reports	Public documentation	High-traffic public docs

---

How to implement "Permanent" URLs in your Agent code

If you choose **Option 1**, your Python function doesn't need to "generate" a URL anymore; it just needs to **construct** it.

Python

```
def get_permanent_url(bucket_name, file_name, region="us-east-1"):
    # Simply construct the string based on the standard AWS format
    return f"https://{bucket_name}.s3.{region}.amazonaws.com/{file_name}"
```

> **Warning:** Be very careful. If your RAG agent ever handles sensitive or personal data, **do not** make the bucket public. Stick to the Pre-signed URLs with a long expiration (e.g., 7 days) instead.

Now that we've cleared up the URL persistence, would you like me to consolidate the "Master Script" that handles the PDF generation, the CJK fonts, and the S3 permanent upload in one go?

yes, please

This master script integrates everything we've discussed. It handles the specific **XeLaTeX** requirements for your Chinese characters paper, manages the **resource paths** for images, and uploads the result to a permanent S3 location.

The Agentic Documentation Master Class

Python

```python
import os
import subprocess
import boto3
from datetime import datetime

class DocumentationAgent:
    def __init__(self, bucket_name, region="us-east-1", cjk_font="Noto Sans CJK SC"):
        self.bucket_name = bucket_name
        self.region = region
        self.cjk_font = cjk_font
        self.s3_client = boto3.client('s3', region_name=region)

    def _generate_yaml_header(self, title):
        """Creates the specialized LaTeX/XeCJK header."""
        return f"""---
title: "{title}"
author: "Generated by Documentation Agent"
date: "{datetime.now().strftime('%Y-%m-%d')}"
mainfont: "DejaVu Sans"
CJKmainfont: "{self.cjk_font}"
header-includes:
  - \\usepackage{{xeCJK}}
  - \\usepackage{{booktabs}}
  - \\usepackage{{longtable}}
geometry: "margin=1in"
---
"""
```

```
    def convert_to_pdf(self, md_body, output_filename, assets_dir="."):
        """Converts Markdown string to PDF using xelatex."""
        title = "Chinese Characters as Living Fossils"
        full_md = self._generate_yaml_header(title) + md_body
        
        md_path = "temp_input.md"
        with open(md_path, "w", encoding="utf-8") as f:
            f.write(full_md)

        # Build the Pandoc command
        cmd = [
            "pandoc", md_path,
            "-o", output_filename,
            "--pdf-engine=xelatex",
            f"--resource-path={assets_dir}",
            "--standalone"
        ]

        try:
            subprocess.run(cmd, check=True)
            return True
        except subprocess.CalledProcessError as e:
            print(f"Conversion Error: {e}")
            return False

    def upload_to_permanent_s3(self, local_file, s3_key):
        """Uploads file and returns the permanent URL (assumes public bucket)."""
        try:
            self.s3_client.upload_file(
                local_file,
                self.bucket_name,
                s3_key,
                ExtraArgs={'ContentType': 'application/pdf'}
            )
            # Construct the permanent URL
            url = f"https://{self.bucket_name}.s3.{self.region}.amazonaws.com/{s3_key}"
            return url
        except Exception as e:
            print(f"Upload Error: {e}")
            return None

# --- Execution ---
# agent = DocumentationAgent(bucket_name="your-permanent-docs-bucket")
# md_content = "# Chapter 1: The Evolution of 山\n\n![Mountain Evolution](mountain.png)"
# if agent.convert_to_pdf(md_content, "living_fossils.pdf", assets_dir="images/"):
#     url = agent.upload_to_permanent_s3("living_fossils.pdf", "research/paper_v1.pdf")
#     print(f"Document live at: {url}")
```

Key Features for Your Paper

1. **UTF-8 Encoding:** The script explicitly opens files with `encoding="utf-8"`, ensuring that your Chinese characters aren't corrupted during the temporary file write stage.
2. **Asset Handling:** The `assets_dir` parameter is crucial. If your RAG agent generates diagrams of character evolutions, this tells Pandoc exactly where to find them.
3. **Automatic Metadata:** It injects the `xeCJK` and `booktabs` packages automatically, so your tables and characters look professional by default.

Final Deployment Checklist

Since you are in **SageMaker Studio**:

- **Bucket Policy:** Ensure your S3 bucket has the "PublicRead" policy we discussed if you want the link to be truly permanent and accessible.
- **Fonts:** If you get a "Font not found" error, run `fc-list :lang=zh` in the SageMaker terminal to find the exact name of the Chinese font installed in your Docker image and update the `cjk_font` variable.
- **Cleaning up:** The script creates a `temp_input.md`. You might want to add `os.remove("temp_input.md")` at the end of the `convert_to_pdf` function to keep your environment clean.

Does this consolidated approach cover all the friction points you were seeing in your agentic RAG pipeline?

Opens in a new window

Google Privacy PolicyGoogle Terms of ServiceYour privacy & Gemini Apps

Gemini may display inaccurate info, including about people, so double-check its responses.