Convert PDF to Audiobook with AI

Google Gemini Transcript & Local Pipeline Guide • June 25, 2026

YOU SAID

Can I upload a pdf and convert it to an audiobook

GEMINI SAID

Yes, you absolutely can! Converting PDFs into audiobooks has become incredibly straightforward thanks to recent leaps in AI text-to-speech (TTS) and natural-sounding voice synthesis.

Depending on how you want to listen, here are the best ways to do it right now:

Dedicated AI Audiobook Converters (Best for a polished experience)

These specialized platforms are designed to handle long-form text. They don't just read the page; they actively clean up the document by skipping junk text like page numbers, headers, footers, and repetitive citations so the narration flows seamlessly.

- **Paper2Audio:** Excellent for complex documents, textbooks, or research papers. It actually summarizes tables and figures instead of trying to read them raw, and it offers a highly generous free tier.
- **Speechify:** The biggest name in this space. It uses incredibly natural, premium AI voices (and even some celebrity voice options). It syncs your reading position across your phone and computer, though the best voices require a premium subscription.
- **Wondercraft / AnyToSpeech:** Great web-based options where you upload a PDF, pick a specific narrator profile (warm, energetic, deep, or professional), and export the final product directly into high-quality MP3 or WAV files to take with you anywhere.

Free & Open-Source Options (For privacy and local control)

If you prefer not to upload your documents to a cloud service or want a completely free desktop setup:

- **NaturalReader (Free Tier):** A great desktop and mobile tool that lets you load PDFs and listen immediately with basic natural voices.
- **Local Python/AI Workflows:** If you enjoy self-hosting or scripting, you can pair a local Python script with tools like **Bark**, **Coqui TTS**, or **Ollama** models to run the text-to-speech conversion completely offline on your own hardware.

💡 **Tips for the Best Conversion Results:**

• **Check for Text Selection:** Make sure your PDF has selectable text. If it's just a scanned image of a document, you will need a tool with built-in OCR (Optical Character Recognition) to recognize the letters before it can read them.

• **Match the Voice to the Genre:** Neutral, stable voices work best for dense technical manuals or documentation, while highly expressive, dynamic voices are much better suited for fiction and storytelling.

YOU SAID

i want to use local ollama

GEMINI SAID

Using local Ollama is a great way to handle the text pre-processing and cleanup phase of building an audiobook. However, it's important to separate the two jobs required for this project:

1. **Text Processing (Ollama's job):** Parsing the raw PDF text, removing messy artifacts (headers, footers, page numbers), and formatting the content into clean, sentence-by-sentence narrative chunks.
2. **Audio Synthesis (TTS engine's job):** Actually converting those clean sentences into highly natural speech. While experimental GGUF-based TTS models exist, dedicated local TTS pipelines produce much faster and more human-sounding audio.

The most efficient, fully local open-source setup divides the workload using a Python script or a pre-built open-source desktop tool.

Method 1: The Pre-Built Desktop Tool (Easiest)

If you want an all-in-one local app that handles the PDF parsing, uses an LLM server for layout optimization, and leverages a blazing-fast local TTS engine, check out **Abogen** on GitHub.

- It uses **Kokoro-82M** locally for highly realistic speech synthesis.
- It hooks into a local inference server via an OpenAI-compatible endpoint (which Ollama provides natively out of the box at `http://localhost:11434/v1`) to use models like **Qwen** to clean up the chapter text.

Method 2: A Custom Python Pipeline (Maximum Control)

If you prefer to orchestrate the pipeline yourself via a custom script, you can build a pipeline that extracts text, passes it to Ollama for conditioning, and then feeds it to a high-quality local TTS engine like Kokoro-onnx or Coqui.

Here is how to design the structural workflow:

1. **Extract Raw PDF Text:** Use a local python library like `pypdf` or `pdfplumber` to extract the raw string data from your PDF file.
2. **Clean and Chunk via Ollama:** Pass the raw text in chunks (e.g., 2000–4000 tokens) to a fast local model like `qwen2.5` or `llama3`. Instruct the model to strip out page headers, running footers, tables of contents, and parenthetical citations while preserving the core narrative text verbatim.
3. **Generate Speech Audio:** Feed the structured, sanitized paragraphs from Ollama into a highly optimized, lightweight local TTS engine. **Kokoro-82M** (via `kokoro-onnx`) or `StyleTTS2` are excellent choices that run locally on consumer GPUs and produce elite audiobook fidelity.
4. **Concatenate Files:** Combine the generated audio segments using a utility like `ffmpeg` to stitch the paragraphs or chapters into unified, portable `.mp3` or `.m4a` files.

Ollama Text Cleaning Prompt Example

When feeding the extracted PDF segments into Ollama to prepare them for your TTS engine, a system prompt like this ensures the model strips out the clutter without altering the author's prose:

```
You are an expert audiobook editor. Your job is to prepare raw extracted PDF
text for a text-to-speech engine.

Tasks:
1. Strip out all page numbers, running headers, footers, and structural
metadata.
2. Silently skip index entries or parenthetical citation keys (e.g., "[1]",
"(Smith, 2023)") that interrupt listening flow.
3. Keep the original narrative prose, dialogue, and author's words EXACTLY as
written. Do not summarize, rephrase, or add commentary.

Output ONLY the clean, sanitized narrative text directly.
```

YOU SAID

yes

GEMINI SAID

Here is a complete, fully local Python automation script. It pairs **Ollama** (for extracting and cleaning text) with **Kokoro**, which is currently the gold-standard open-source engine for fast, highly realistic, completely offline voice synthesis.

Prerequisites

You will need to install the required Python libraries. Open your terminal and run:

```
pip install pypdf requests kokoro soundfile torch
```

Note: If you are on Linux or Mac, you may also need to install `espeak-ng` via your package manager (e.g., `sudo apt-get install espeak-ng` or `brew install espeak-ng`) as it is used by Kokoro for phonetic rendering.

Make sure your local Ollama server is running. Ensure you have downloaded a fast, smart text model like Qwen or Llama by running:

```
ollama run qwen2.5:7b
```

The Python Script (`pdf_to_audiobook.py`)

Create a script using the code below. It loops through your PDF, hands the raw text to Ollama to strip out page layout junk, and feeds the cleaned prose to Kokoro to generate a flawless local audiobook.

```
import os
import requests
from pypdf import PdfReader
from kokoro import KPipeline
import soundfile as sf
import numpy as np

# --- CONFIGURATION ---
PDF_PATH = "your_book.pdf"       # Path to your local PDF file
OUTPUT_AUDIO = "audiobook.wav"   # Name of the output audio file
OLLAMA_MODEL = "qwen2.5:7b"     # The local Ollama model you want to use
KOKORO_VOICE = "am_michael"      # Try "af_sarah" (female) or
"am_michael" (male)
# ---------------------

def clean_text_with_ollama(raw_text):
    """Sends raw text chunks to local Ollama to remove PDF headers, footers,
and citation artifacts."""
    url = "http://localhost:11434/api/generate"
    system_prompt = (
        "You are an expert audiobook editor. Clean up this raw text extracted
from a PDF. "
        "Remove running headers, footers, page numbers, and inline citation
brackets (like [1] or (Smith, 2021)). "
        "Do NOT summarize, rephrase, or change the narrative text. Return ONLY
the clean narrative prose."
    )

    payload = {
        "model": OLLAMA_MODEL,
        "prompt": f"{system_prompt}\n\nRaw Text:\n{raw_text}",
        "stream": False
    }
    try:
        response = requests.post(url, json=payload)
        return response.json().get("response", "").strip()
    except Exception as e:
        print(f"Ollama connection error: {e}. Falling back to raw text.")
        return raw_text

def main():
    if not os.path.exists(PDF_PATH):
        print(f"Error: Could not find PDF at '{PDF_PATH}'")
        return

    print("📖 Initializing text extraction and local TTS engine...")
    reader = PdfReader(PDF_PATH)
```

```
    # Lang code 'a' stands for American English ('b' for British English)
    tts_pipeline = KPipeline(lang_code='a')

    all_audio_chunks = []
    total_pages = len(reader.pages)
    print(f"Processing {total_pages} pages total.")

    for i, page in enumerate(reader.pages):
        print(f"\n--- Processing Page {i+1} / {total_pages} ---")
        raw_text = page.extract_text()
        if not raw_text.strip():
            continue

        print("🧼 Cleaning layout artifacts via Ollama...")
        cleaned_text = clean_text_with_ollama(raw_text)
        if not cleaned_text:
            continue

        print("🎙️ Synthesizing human-like speech locally...")
        # Kokoro processes the text and splits long phrases smoothly under the
hood
        generator = tts_pipeline(cleaned_text, voice=KOKORO_VOICE, speed=1.0)
        for _, _, audio in generator:
            all_audio_chunks.append(audio)

    if all_audio_chunks:
        print(f"\n💾 Combining and saving audiobook to {OUTPUT_AUDIO}...")
        # Combine all audio NumPy arrays along the timeline
        final_audio = np.concatenate(all_audio_chunks)
        # Kokoro models output audio at a native 24000 Hz sample rate
        sf.write(OUTPUT_AUDIO, final_audio, 24000)
        print("🎉 Finished! Your fully local audiobook is ready.")
    else:
        print("❌ No audio chunks were generated.")

if __name__ == "__main__":
    main()
```