# YouTube Shorts Subtitles App (Next.js) — with Komika Axis font

> Note: You wrote "Komic Axis" — the widely used font is **Komika Axis** (aka *KOMIKAX_.ttf*). This project burns captions using that font and also exports SRT/WebVTT.

## What you get

- Upload a vertical video (9:16)
- Auto-transcribe to **English** via OpenAI **GPT-4o Transcribe**
- Caption editor (timeline list)
- Export **SRT**/**WebVTT** files
- **Hard-sub** burn-in using **ffmpeg.wasm** with *Komika Axis*

> ⚖️ **Font licensing** varies by source. Many aggregators mark Komika Axis as free or free-for-commercial-use, but some list it as personal-use only. Verify the license in the font ZIP you download before commercial release.

---

## Quick start

```
# 1) Create app
npx create-next-app@latest shorts-subtitles --ts --eslint --app --src-dir --
tailwind
cd shorts-subtitles

# 2) Install deps
npm i @ffmpeg/ffmpeg @ffmpeg/util zod react-hook-form openai

# 3) Add env (server only)
# .env.local
OPENAI_API_KEY=sk-...  # keep server-side only

# 4) Add files from this doc into your project (paths below)

# 5) Place the font
# Put KOMIKAX_.ttf at: public/fonts/KOMIKAX_.ttf

# 6) Run
npm run dev
```

---

## /app/api/transcribe/route.ts (server, Edge disabled)

```
// app/api/transcribe/route.ts
import { NextRequest, NextResponse } from "next/server";
import OpenAI from "openai";

export const runtime = "nodejs"; // we need Node for file handling

const client = new OpenAI({ apiKey: process.env.OPENAI_API_KEY });

export async function POST(req: NextRequest) {
  try {
    const form = await req.formData();
    const file = form.get("file") as File | null;
    if (!file) return NextResponse.json({ error: "No file" }, { status:
400 });

    // Send to OpenAI GPT-4o Transcribe (preferred over whisper-1)
    const result = await client.audio.transcriptions.create({
      file,
      model: "gpt-4o-transcribe", // see docs
      language: "en",
      response_format: "verbose_json", // word-level timestamps
      timestamp_granularities: ["word"],
      temperature: 0.0,
    } as any);

    return NextResponse.json(result);
  } catch (err: any) {
    console.error(err);
    return NextResponse.json({ error: err?.message || "Transcription
failed" }, { status: 500 });
  }
}
```

If you prefer SRT directly from the API, change `response_format: "srt"` and parse. We default to `verbose_json` to compute smart line breaks for Shorts.

---

## /src/app/globals.css — add font-face and subtitle style

```
@tailwind base;
@tailwind components;
@tailwind utilities;

/* Komika Axis font */
@font-face {
  font-family: 'Komika Axis';
```

```
  src: url('/fonts/KOMIKAX_.ttf') format('truetype');
  font-weight: 400;
  font-style: normal;
  font-display: swap;
}

/* Subtitle text look for preview (burn-in uses ASS styling below) */
.subtitle-preview {
  font-family: 'Komika Axis', system-ui, sans-serif;
  letter-spacing: 0.5px;
  -webkit-text-stroke: 2px rgba(0,0,0,0.9);
  text-shadow:
    0 0 2px #000, 0 0 6px #000,
    0 0 10px rgba(0,0,0,0.8);
}
```

---

## /src/app/page.tsx (frontend UI + ffmpeg burn-in)

```
'use client';
import React, { useEffect, useMemo, useRef, useState } from 'react';
import { createFFmpeg, fetchFile } from '@ffmpeg/ffmpeg';
import { z } from 'zod';

const ffmpeg = createFFmpeg({ log: true });

type Word = { start: number; end: number; text: string };

// Chunk words into 1–2 lines suited for Shorts
function wordsToCues(words: Word[], maxChars = 38) {
  const cues: { start: number; end: number; text: string }[] = [];
  let buff: Word[] = [];
  const push = () => {
    if (!buff.length) return;
    const start = buff[0].start;
    const end = buff[buff.length - 1].end;
    const text = buff.map(w => w.text).join(' ');
    cues.push({ start, end, text });
    buff = [];
  };
  for (const w of words) {
    const tentative = (buff.map(b => b.text).join(' ') + ' ' +
w.text).trim();
    if (tentative.length > maxChars) push();
    buff.push(w);
    if (tentative.endsWith('.') || tentative.endsWith('!') ||
tentative.endsWith('?')) push();
  }
  push();
```

```
  // pad short durations
  for (const c of cues) if (c.end - c.start < 0.9) c.end = c.start + 0.9;
  return cues;
}

function toSRT(cues: { start: number; end: number; text: string }[]) {
  const fmt = (t: number) => new Date(t * 1000).toISOString().slice(11,
23).replace('.', ',');
  return cues.map((c, i) => `${i + 1}\n${fmt(c.start)} --> ${fmt(c.end)}\n$
{c.text}\n`).join('\n');
}

function toVTT(cues: { start: number; end: number; text: string }[]) {
  const fmt = (t: number) => new Date(t * 1000).toISOString().slice(11, 23);
  return 'WEBVTT\n\n' + cues.map(c => `${fmt(c.start)} --> ${fmt(c.end)}\n$
{c.text}\n`).join('\n');
}

function toASS(cues: { start: number; end: number; text: string }[]) {
  const ts = (t: number) => {
    const h = Math.floor(t / 3600).toString().padStart(1, '0');
    const m = Math.floor((t % 3600) / 60).toString().padStart(2, '0');
    const s = (t % 60).toFixed(2).padStart(5, '0');
    return `${h}:${m}:${s}`;
  };
  const header = `\[Script Info]\nScriptType: v4.00+\nPlayResX:
1080\nPlayResY: 1920\nScaledBorderAndShadow: yes\n\n\[V4+ Styles]\nFormat:
Name, Fontname, Fontsize, PrimaryColour, SecondaryColour, OutlineColour,
BackColour, Bold, Italic, Underline, StrikeOut, ScaleX, ScaleY, Spacing,
Angle, BorderStyle, Outline, Shadow, Alignment, MarginL, MarginR,
MarginV\nStyle: AXIS,Komika Axis,
72,&H00FFFFFF,&H000000FF,&H00000000,&H7F000000,-1,0,0,0,100,100,0,0,1,6,0,2,50,50,120\n\n\
[Events]\nFormat: Layer, Start, End, Style, Name, MarginL, MarginR, MarginV,
Effect, Text\n`;
  const events = cues.map(c => `Dialogue: 0,${ts(c.start)},$
{ts(c.end)},AXIS,,0,0,0,,{\\an2} ${c.text.replace(/\n/g, ' ')}\n`).join('');
  return header + events;
}

export default function Page() {
  const [videoFile, setVideoFile] = useState<File [] null>(null);
  const [videoUrl, setVideoUrl] = useState<string [] null>(null);
  const [cues, setCues] = useState<{ start: number; end: number; text:
string }[]>([]);
  const [busy, setBusy] = useState<string [] null>(null);

  useEffect(() => {
    if (!videoFile) return;
    const url = URL.createObjectURL(videoFile);
    setVideoUrl(url);
    return () => URL.revokeObjectURL(url);
```

```
  }, [videoFile]);

  async function transcribe() {
    if (!videoFile) return;
    setBusy('Transcribing…');
    const body = new FormData();
    body.append('file', videoFile);
    const res = await fetch('/api/transcribe', { method: 'POST', body });
    if (!res.ok) throw new Error('Transcription failed');
    const data = await res.json();
    // Expect verbose_json with word timestamps
    const words: Word[] = (data?.words || data?.segments?.flatMap((s: any)
=> s.words) || []).map((w: any) => ({
      start: w.start ?? w.timestamp?.start ?? 0,
      end: w.end ?? w.timestamp?.end ?? 0,
      text: (w.text || w.word || '').trim(),
    })).filter((w: Word) => w.text);
    setCues(wordsToCues(words));
    setBusy(null);
  }

  async function burnIn() {
    if (!videoFile || !cues.length) return;
    setBusy('Preparing ffmpeg… (first run can take ~30–60s)');
    if (!ffmpeg.isLoaded()) await ffmpeg.load();

    setBusy('Writing files…');
    ffmpeg.FS('writeFile', 'input.mp4', await fetchFile(videoFile));
    ffmpeg.FS('writeFile', 'subs.ass', new
TextEncoder().encode(toASS(cues)));
    // provide font to libass
    const fontResp = await fetch('/fonts/KOMIKAX_.ttf');
    ffmpeg.FS('writeFile', 'KOMIKAX_.ttf', await fontResp.arrayBuffer());

    setBusy('Burning subtitles…');
    // Use -vf subtitles with font provider (ass autocaches attached fonts).
Some builds need FONTCONFIG.
    await ffmpeg.run(
      '-i', 'input.mp4',
      '-vf', "ass=subs.ass",
      '-c:v', 'libx264', '-preset', 'veryfast', '-crf', '20',
      '-c:a', 'copy',
      'output.mp4'
    );

    const data = ffmpeg.FS('readFile', 'output.mp4');
    const url = URL.createObjectURL(new Blob([data.buffer], { type: 'video/
mp4' }));
    const a = document.createElement('a');
    a.href = url; a.download = 'short-with-subtitles.mp4'; a.click();
    URL.revokeObjectURL(url);
```

```
    setBusy(null);
  }

  function download(type: 'srt' | 'vtt') {
    const text = type === 'srt' ? toSRT(cues) : toVTT(cues);
    const blob = new Blob([text], { type: 'text/plain;charset=utf-8' });
    const url = URL.createObjectURL(blob);
    const a = document.createElement('a');
    a.href = url; a.download = `captions.${type}`; a.click();
    URL.revokeObjectURL(url);
  }

  return (
    <main className="min-h-screen bg-slate-50">
      <div className="max-w-5xl mx-auto p-4 sm:p-8">
        <h1 className="text-3xl font-bold tracking-tight">Shorts Subtitles — Komika Axis</h1>
        <p className="text-sm text-slate-600 mt-1">Upload, auto-transcribe to English, edit, export SRT/VTT, or burn-in.</p>

        <div className="mt-6 grid grid-cols-1 lg:grid-cols-2 gap-6">
          <section className="p-4 rounded-2xl bg-white shadow">
            <label className="block text-sm font-medium">Video (9:16 preferred)</label>
            <input className="mt-2" type="file" accept="video/*" onChange={e => setVideoFile(e.target.files?.[0] || null)} />
            {videoUrl && (
              <div className="mt-4">
                <video src={videoUrl} className="rounded-xl w-full" controls playsInline />
              </div>
            )}
            <div className="mt-4 flex gap-2">
              <button onClick={transcribe} disabled={!videoFile || !!busy} className="px-4 py-2 rounded-lg bg-black text-white disabled:opacity-50">Transcribe</button>
              <button onClick={() => download('srt')} disabled={!cues.length} className="px-4 py-2 rounded-lg border">Download SRT</button>
              <button onClick={() => download('vtt')} disabled={!cues.length} className="px-4 py-2 rounded-lg border">Download VTT</button>
              <button onClick={burnIn} disabled={!cues.length || !!busy} className="px-4 py-2 rounded-lg bg-emerald-600 text-white disabled:opacity-50">Burn-In (ASS)</button>
            </div>
            {busy && <p className="mt-3 text-sm text-slate-600">{busy}</p>}
          </section>

          <section className="p-4 rounded-2xl bg-white shadow">
            <h2 className="font-semibold">Captions</h2>
            {!cues.length && <p className="text-sm text-slate-500 mt-2">No captions yet. Click Transcribe.</p>}
```

```
            <ul className="mt-3 space-y-2 max-h-[520px] overflow-auto">
              {cues.map((c, i) => (
                <li key={i} className="border rounded-lg p-2">
                  <div className="text-[10px] text-
slate-500">{c.start.toFixed(2)} → {c.end.toFixed(2)}</div>
                  <input
                    className="w-full mt-1 subtitle-preview text-xl"
                    value={c.text}
                    onChange={e => setCues(prev => prev.map((x, idx) => idx
=== i ? { ...x, text: e.target.value } : x))}
                  />
                </li>
              ))}
            </ul>
          </section>
        </div>

        <p className="text-xs text-slate-500 mt-6">Tip: For crisp Shorts,
keep each caption under ~38 characters and ~1.0–1.5s on screen.</p>
      </div>
    </main>
  );
}
```

---

## Notes

- **Transcription model**: this uses **GPT-4o Transcribe** as per OpenAI's audio docs. You can swap to another provider if you prefer.
- **Font embedding** for burn-in: we write `KOMIKAX_.ttf` into ffmpeg's virtual FS so libass can render the `ASS` style named `Komika Axis`.
- **Performance**: ffmpeg.wasm is CPU-heavy in browsers. For longer videos, offload burn-in to a server (ffmpeg CLI) and keep the browser for SRT/VTT only.
- **Shorts safe area**: our ASS style uses `Alignment=2` (bottom center) and a large size; tweak `MarginV` if your UI overlays cover captions.

---

## Optional: server-side burn-in (faster)

On your server, run:

```
ffmpeg -i input.mp4 -vf "ass=subs.ass" -c:v libx264 -crf 20 -preset veryfast
-c:a copy output.mp4
```

## Optional: Komika Axis as CSS overlay (no burn-in)

You can also render live captions in the player using `track kind="subtitles"` (WebVTT) and CSS with `.subtitle-preview` for the font look.

---

## One-click Deploy (Vercel)

You can deploy this as a standard Next.js app. Two options:

### A) Import from GitHub (recommended)

1) Create a new GitHub repo and paste this project's files. 2) Go to Vercel → **Add New Project** → import the repo. 3) In **Environment Variables**, add: - `OPENAI_API_KEY` = your key 4) Hit **Deploy**. That's it.

### B) Vercel CLI

```
npm i -g vercel
vercel login
vercel init   # if needed
vercel link   # inside project folder
vercel env add OPENAI_API_KEY  # paste your key when prompted
vercel --prod
```

### vercel.json (optional, sets serverless region)

Singapore is usually low-latency for India users. You can omit this file and Vercel will choose automatically.

```
{
  "functions": {
    "api/**": { "runtime": "nodejs20.x", "memory": 1024,
"maxDuration": 60 }
  },
  "regions": ["sin1"]
}
```

### Notes for Vercel

- The API route `/api/transcribe` already sets `export const runtime = "nodejs";` so it will run as a Node serverless function.
- No additional build step required beyond the default `next build`.
- Keep your `OPENAI_API_KEY` on the **server** (do not expose on client).

---

## Sample video (9:16) + “Load sample” button

Add a short vertical clip so you can test in seconds: 1) Put a ~3–10s MP4 at `public/sample.mp4` (any CC0/your own clip). 2) Update the UI to add a **Load sample** button that fetches this file and sets it as the working video.

Append this patch to `/src/app/page.tsx` (search for the button row and add the new button + helper):

```
// Add this helper inside the component
async function loadSample() {
  const res = await fetch('/sample.mp4');
  const blob = await res.blob();
  const f = new File([blob], 'sample.mp4', { type: 'video/mp4' });
  setVideoFile(f);
}
```

Then in the button toolbar (next to Transcribe/Download/Burn-In), add:

```
<button onClick={loadSample} className="px-4 py-2 rounded-lg border">Load
sample</button>
```

> Tip: If your sample has non-English speech, the server still forces English output; switch to auto-detect by removing `language: "en"` or set to your source language for better accuracy.

---

## ffmpeg-generated dummy vertical sample (optional)

If you don’t have a clip handy, you can generate an MP4 locally:

```
# Creates a 5-second 1080x1920 silent clip with “TEST” text
ffmpeg -f lavfi -i color=c=black:s=1080x1920:d=5
  -vf "drawtext=fontfile=./public/fonts/KOMIKAX_.ttf:text='TEST':x=(w-tw)/
2:y=h*0.75:fontsize=96:fontcolor=white:shadowcolor=black:shadowx=3:shadowy=3"
  -r 30 public/sample.mp4 -y
```

---

## Production hard-sub (server) – optional API

For longer videos, add a server route that burns subtitles faster than ffmpeg.wasm:

`/app/api/burn/route.ts`

```ts
import { NextRequest, NextResponse } from 'next/server';
import { spawn } from 'node:child_process';

export const runtime = 'nodejs';

export async function POST(req: NextRequest) {
  const form = await req.formData();
  const file = form.get('file') as File | null;
  const ass = form.get('ass') as string | null;
  if (!file || !ass) return NextResponse.json({ error: 'Missing file or
ASS' }, { status: 400 });

  // Save inputs to /tmp
  const buff = Buffer.from(await file.arrayBuffer());
  const fs = await import('node:fs/promises');
  await fs.writeFile('/tmp/in.mp4', buff);
  await fs.writeFile('/tmp/subs.ass', ass);

  // Run ffmpeg CLI
  const args = ['-y','-i','/tmp/in.mp4','-vf','ass=/tmp/subs.ass','-
c:v','libx264','-preset','veryfast','-crf','20','-c:a','copy','/tmp/
out.mp4'];
  await new Promise((resolve, reject) => {
    const p = spawn('ffmpeg', args);
    p.on('error', reject);
    p.on('close', code => code === 0 ? resolve(null) : reject(new
Error('ffmpeg failed')));
  });

  const data = await fs.readFile('/tmp/out.mp4');
  return new NextResponse(data, { headers: { 'Content-Type': 'video/
mp4' } });
}
```

> On Vercel, `ffmpeg` is available in the build image; for serverless at runtime, consider bundling ffmpeg static or using a background job. If serverless limits bite, run this on a Node server/Edge function compatible host.

---

## QA checklist before going live

- [ ] Test with `public/sample.mp4` → **Transcribe** → **Burn-In** → gets `short-with-subtitles.mp4`
- [ ] Verify **Komika Axis** font license in your font ZIP and keep the TTF self-hosted
- [ ] Confirm captions fit Shorts safe area (no overlap with app UI)
- [ ] Set `regions` in `vercel.json` if your audience is primarily in India/South-Asia
- [ ] Add rate-limit on `/api/transcribe` (e.g., IP + file size guard)