```
// server.js (Node 18+ recommended)
import express from 'express';
import fetch from 'node-fetch'; // or global fetch in Node 18+
import dotenv from 'dotenv';
dotenv.config();

const app = express();
app.use(express.json({ limit: '10mb' })); // payload limit (adjust carefully)
const PORT = process.env.PORT || 3000;
const API_KEY = process.env.GEMINI_API_KEY;
if (!API_KEY) {
  console.error('Set GEMINI_API_KEY in env');
  process.exit(1);
}

app.post('/analyze', async (req, res) => {
  try {
    const { frames = [], focus = 'auto' } = req.body;
    if (!Array.isArray(frames) || frames.length === 0) {
      return res.status(400).json({ error: 'No frames provided' });
    }

    // Limit number of frames server-side to 6-8 to avoid payload bloat
    const limited = frames.slice(0, 8);

    // Build the multimodal payload per Gemini generateContent format (REST)
    const model = 'gemini-2.5-flash-preview-2025'; // check latest model name
    const url =
`https://generativelanguage.googleapis.com/v1beta/models/${model}:generateContent?key=${A
PI_KEY}`;

    // Build parts: each image as an inlineData part, plus a text instruction part
    const imageParts = limited.map(b64 => ({ inlineData: { mimeType: 'image/jpeg', data: b64 }
}));
    const instructionText = `Analyze these frames for deepfake artifacts. Focus: ${focus}.
Return a single JSON object with keys: result ('REAL' or 'DEEPFAKE'), reason (max 2
sentences), score (integer 50-99).`;

    const payload = {
      contents: [
        {
          parts: [
            ...imageParts,
            { text: instructionText }
```

```
        ]
       }
      ],
      generationConfig: {
       // Ask for JSON output via responseSchema, but some REST endpoints
       // expect "responseMimeType" or "responseSchema". We'll request JSON mode:
       responseMimeType: "application/json"
      },
      // Optionally add systemInstruction to constrain style
      systemInstruction: {
       parts: [{ text: "You are a Deepfake Video Analysis Engine. Respond only with a single
JSON object." }]
      }
    };

    const apiResp = await fetch(url, {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify(payload)
    });

    if (!apiResp.ok) {
      const t = await apiResp.text();
      console.error('Gemini API failed', apiResp.status, t);
      return res.status(502).json({ error: 'AI provider error', details: t });
    }

    const json = await apiResp.json();

    // Extract text result (depends on model output structure)
    const jsonText = json?.candidates?.[0]?.content?.parts?.[0]?.text;
    if (!jsonText) {
      return res.status(502).json({ error: 'No JSON from model', raw: json });
    }

    // This is expected to be a JSON string — parse it safely
    let analysis;
    try {
      analysis = JSON.parse(jsonText);
    } catch (e) {
      // If not valid JSON, try to extract substring or fallback
      return res.status(502).json({ error: 'Model returned invalid JSON', rawText: jsonText });
    }
```

```
    // Normalize/validate fields
    const result = {
      result: analysis.result || 'UNKNOWN',
      reason: analysis.reason || '',
      score: Math.max(0, Math.min(100, parseInt(analysis.score || 0)))
    };

    return res.json(result);

  } catch (err) {
    console.error(err);
    return res.status(500).json({ error: err.message || 'internal error' });
  }
});

app.listen(PORT, () => console.log(`Server listening on ${PORT}`));
```