Claude Code Debug Prompt: Image Enhancement Not Working

Summary of Current Status

✅ FIXED Issues

- **HEIC Upload**: Now works perfectly using `heic-to` library (libheif 1.21.2 with H.265 support)
- **Admin Dashboard Photos**: Display correctly now
- Console shows successful conversion: `[HEIC] Conversion successful, output size: 2402519`

❌ Current Issues

1. **Manual Enhancement Not Working**
   - User clicks “Run Enhancement” button with settings selected
   - Button appears to “load” (something is happening)
   - But enhancement never completes - photo remains “Not yet enhanced”
   - No error messages shown to user

2. **Auto Enhancement Not Implemented**
   - User expected photos to be automatically enhanced upon submission using base prompts
   - Currently there is NO auto-enhancement - all enhancement must be triggered manually

3. **Feature Request: Editable Prompts**
   - User wants ability to edit the base prompt when running manual enhancement
   - User wants a Settings page to permanently edit base prompts for all future enhancements

---

Code Analysis Needed

Enhancement API Route (`/api/photos/[id]/enhance/route.ts`)

```
import { NextResponse } from "next/server";
import { prisma } from "@/lib/db";
import { getServerSession } from "next-auth";
import { authOptions } from "@/lib/auth-options";
import { ROOM_PROMPTS, INTENSITY_MODIFIERS } from "@/lib/enhancement-prompts";
import { getFileUrl } from "@/lib/s3";

export const dynamic = "force-dynamic";

export async function POST(
  request: Request,
  { params }: { params: { id: string } }
) {
  const session = await getServerSession(authOptions);
  if (!session) {
    return NextResponse.json({ error: "Unauthorized" }, { status: 401 });
  }

  try {
    const data = await request.json();
    const {
      intensity = "Moderate",
      skyReplacement = false,
      bedFixing = false,
      windowRecovery = false,
      brightness = false,
      perspective = false,
      reflection = false,
      additionalNotes = ""
    } = data;

    // Get photo with original URL
    const photo = await prisma.photo.findUnique({
      where: { id: params.id },
      include: {
        enhancementVersions: {
          orderBy: { versionNumber: 'desc' },
          take: 1
        }
      }
    });

    if (!photo) {
      return NextResponse.json({ error: "Photo not found" }, { status: 404 });
    }

    // Always use ORIGINAL photo URL, never the enhanced one
    const originalUrl = await getFileUrl(photo.originalUrl, true);

    // Build the prompt
    const roomKey = photo.subCategory || photo.roomCategory;
    let prompt = ROOM_PROMPTS[roomKey] || ROOM_PROMPTS[photo.roomCategory] || ROOM_PRO
MPTS["Kitchen"];
    prompt += INTENSITY_MODIFIERS[intensity] || INTENSITY_MODIFIERS["Moderate"];

    // Add enhancement toggles
    const toggles: string[] = [];
    if (skyReplacement) toggles.push("Apply sky replacement with pleasant blue sky");
    if (bedFixing) toggles.push("Smooth and fix bed linens to appear crisp and invit-
ing");
    if (windowRecovery) toggles.push("Recover and reveal exterior view through win-
dows");
```

```
    if (brightness) toggles.push("Increase overall brightness and reduce shadows");
    if (perspective) toggles.push("Correct perspective and straighten vertical lines")
;
    if (reflection) toggles.push("Remove photographer reflections from mirrors and
glass");

    if (toggles.length > 0) {
      prompt += `\n\nADDITIONAL ENHANCEMENT INSTRUCTIONS:\n${toggles.join('\n')}`;
    }

    if (additionalNotes) {
      prompt += `\n\nADMIN NOTES:\n${additionalNotes}`;
    }

    // Stream the LLM response
    const response = await fetch('https://apps.abacus.ai/v1/chat/completions', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
        'Authorization': `Bearer ${process.env.ABACUSAI_API_KEY}`
      },
      body: JSON.stringify({
        model: 'gpt-5.1',
        messages: [
          {
            role: 'user',
            content: [
              {
                type: 'image_url',
                image_url: { url: originalUrl }
              },
              {
                type: 'text',
                text: prompt
              }
            ]
          }
        ],
        modalities: ['image'],
        image_config: {
          num_images: 1,
          aspect_ratio: photo.orientation === 'portrait' ? '3:4' : '4:3'
        },
        stream: true,
        max_tokens: 1000
      })
    });

    if (!response.ok) {
      const errorText = await response.text();
      console.error('LLM API error:', errorText);
      return NextResponse.json({ error: "Enhancement API failed" }, { status: 500 });
    }

    // Stream the response back to client
    const stream = new ReadableStream({
      async start(controller) {
        const reader = response.body?.getReader();
        const decoder = new TextDecoder();
        const encoder = new TextEncoder();
        let partialRead = '';
        let enhancedImageUrl = '';
```

```
        try {
          while (reader) {
            const { done, value } = await reader.read();
            if (done) break;

            partialRead += decoder.decode(value, { stream: true });
            const lines = partialRead.split('\n');
            partialRead = lines.pop() || '';

            for (const line of lines) {
              if (line.startsWith('data: ')) {
                const data = line.slice(6);
                if (data === '[DONE]') {
                  // Enhancement complete - save to database
                  if (enhancedImageUrl) {
                    const versionNumber = (photo?.enhancementVersions?.[0]?.version-
Number || 0) + 1;

                    await prisma.enhancementVersion.create({
                      data: {
                        photoId: params.id,
                        versionNumber,
                        enhancedUrl: enhancedImageUrl,
                        intensity,
                        skyReplacement,
                        bedFixing,
                        windowRecovery,
                        brightness,
                        perspective,
                        reflection,
                        additionalNotes: additionalNotes || null
                      }
                    });

                    await prisma.photo.update({
                      where: { id: params.id },
                      data: {
                        enhancedUrl: enhancedImageUrl,
                        status: 'Enhanced'
                      }
                    });

                    const finalData = JSON.stringify({
                      status: 'completed',
                      enhancedUrl: enhancedImageUrl,
                      versionNumber
                    });
                    controller.enqueue(encoder.encode(`data: ${finalData}\n\n`));
                  }
                  return;
                }

                try {
                  const parsed = JSON.parse(data);
                  // Check for image in response
                  const images = parsed.choices?.[0]?.delta?.images ||
parsed.choices?.[0]?.message?.images;
                  if (images && images.length > 0) {
                    enhancedImageUrl = images[0]?.image_url?.url || images[0]?.url ||
'';
                  }

                  // Send progress update
```

```
                  const progressData = JSON.stringify({
                    status: 'processing',
                    message: 'Enhancing image...'
                  });
                  controller.enqueue(encoder.encode(`data: ${progressData}\n\n`));
                } catch {
                  // Skip invalid JSON
                }
              }
            }
          }
        } catch (error) {
          console.error('Stream error:', error);
          controller.error(error);
        } finally {
          controller.close();
        }
      }
    });

    return new Response(stream, {
      headers: {
        'Content-Type': 'text/event-stream',
        'Cache-Control': 'no-cache',
        'Connection': 'keep-alive'
      }
    });
  } catch (error) {
    console.error('Enhancement error:', error);
    return NextResponse.json({ error: "Enhancement failed" }, { status: 500 });
  }
}
```

Client-Side Enhancement Handler (`admin/submissions/[id]/page.tsx`)

```
const handleEnhance = async () => {
  if (!selectedPhoto) return;
  setEnhancing(true);

  try {
    const response = await fetch(`/api/photos/${selectedPhoto.id}/enhance`, {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify(enhanceSettings)
    });

    const reader = response.body?.getReader();
    const decoder = new TextDecoder();
    let partialRead = '';

    while (reader) {
      const { done, value } = await reader.read();
      if (done) break;

      partialRead += decoder.decode(value, { stream: true });
      const lines = partialRead.split('\n');
      partialRead = lines.pop() || '';

      for (const line of lines) {
        if (line.startsWith('data: ')) {
          try {
            const data = JSON.parse(line.slice(6));
            if (data?.status === 'completed' && data?.enhancedUrl) {
              // Refresh submission to get updated photo
              await fetchSubmission();
            }
          } catch {
            // Skip invalid JSON
          }
        }
      }
    }
  } catch (error) {
    console.error('Enhancement failed:', error);
    alert('Enhancement failed. Please try again.');
  } finally {
    setEnhancing(false);
  }
};
```

Possible Issues to Investigate

1. LLM API Response Format

- The code expects images in: `parsed.choices?.[0]?.delta?.images` or `parsed.choices?.[0]?.message?.images`
- Is this the correct response format for Abacus AI's image generation API?
- The model used is `gpt-5.1` - is this correct for image generation?

2. Stream Parsing

- Is the SSE stream being parsed correctly?
- Are we correctly detecting the `[DONE]` signal?
- Is `enhancedImageUrl` being extracted from the right place in the response?

3. API Key / Authentication

- The `ABACUSAI_API_KEY` environment variable IS set (verified)
- But is it valid? Is the API returning an error silently?

4. Image URL Format

- `getFileUrl(photo.originalUrl, true)` is used to get the image URL
- Is this returning a valid, accessible URL for the LLM to fetch?

5. Missing Error Handling

- No error is shown to the user if enhancement fails silently
- Need better logging and user feedback

Questions for You (Claude Code)

1. **What is the correct API format for Abacus AI image generation?**
   - Is `modalities: ['image']` correct?
   - What model should be used for image enhancement/generation?
   - What does the response format look like for image generation?

2. **Why might the enhancement be "loading" but never completing?**
   - Stream not being read correctly?
   - Response format mismatch?
   - API error being swallowed?

3. **How should auto-enhancement be implemented?**
   - Should it happen in the `/api/submissions` POST route after photos are saved?
   - Should it be a background job/queue?
   - Should there be a "status" indicator showing auto-enhancement in progress?

4. **What's the best way to implement editable prompts?**
   - Store prompts in database instead of hardcoded in code?
   - Create a Settings page for admin to edit prompts?
   - Allow per-photo prompt editing before enhancement?

What I Need From You

1. **Diagnose the enhancement failure** - Why is "Run Enhancement" not working?
2. **Fix the LLM API call** - Correct format for Abacus AI image generation
3. **Add proper error handling and logging** - Surface errors to console and user
4. **Implement editable prompts feature** - Textarea to edit prompt before enhancement
5. **Optionally: Design auto-enhancement approach** - If feasible

Please provide exact code changes for DeepAgent to implement.

🔴 CRITICAL: Server Logs Show Stream Errors

The production server logs reveal the actual error:

```
Stream error: TypeError: terminated
    at Fetch.onAborted (node:internal/deps/undici/undici:11124:53)
    ...
  [cause]: SocketError: other side closed
      ...
      code: 'UND_ERR_SOCKET',
      socket: {
        localAddress: '100.64.0.46',
        localPort: 47610,
        remoteAddress: '104.20.35.251',
        remotePort: 443,
        ...
        bytesWritten: 3179,
        bytesRead: 929
      }

Error: failed to pipe response
```

This tells us:
1. The connection to the LLM API (`104.20.35.251:443` - Cloudflare/Abacus AI) is being **terminated by the remote server**
2. Only 929 bytes were received before the connection closed
3. This is likely an **API error or authentication issue**, not a code bug

Possible causes:
1. Invalid API key
2. Wrong model name (`gpt-5.1` may not exist for image generation)
3. Incorrect API format for image generation
4. Rate limiting
5. The `modalities: ['image']` parameter may not be supported

Key question for you (Claude Code):
- What is the CORRECT way to call Abacus AI / RouteLLM API for image generation/enhancement?
- Is `gpt-5.1` a valid model for image generation?
- What should the request format look like?

The image at `104.20.35.251` is being sent but the server immediately closes the connection after sending very little data (929 bytes). This suggests the API is rejecting the request early.