ChatGPT

Nanobanana MCP 응답 크기 오류 문제 해결

문제 현상 (Error Symptom)

NanoBanana MCP 서버에서 이미지를 생성하려 할 때 `"MCP tool 'nanobanana_generate' response ... exceeds maximum allowed tokens (25000)"` 라는 오류가 발생했습니다. 이는 Claude Code 환경에서 **MCP 툴의 응답 크기가 너무 커서** 발생하는 에러입니다. 구체적으로, 강아지가 강을 건너는 이미지를 생성하려고 할 때 **응답 데이터의 용량이 과도하게 커져** 요청이 거부된 것입니다.

문제 원인 (Root Cause)

이 문제의 근본 원인은 **이미지 생성 결과를 MCP 서버가 처리하는 방식**에 있습니다. 현재 Nanobanana MCP 서버는 이미지를 생성한 후, **그 이미지를 base64로 인코딩한 거대한 문자열 데이터를 응답에 포함**시킵니다 [1]. Base64로 인코딩된 이미지 데이터는 용량이 매우 크기 때문에 Claude Code가 한 번에 처리할 수 있는 토큰 한도를 넘어버립니다.

추가로 분석해보면, 코드상으로는 `generate_image.py`와 `edit_image.py`에서 결과를 반환할 때 `image_data["base64"]` 필드에 **대용량의 base64 문자열**을 포함하고 있었습니다. 또한 `gemini_client.py`에서는 `result["images"]` 리스트 각 항목에 `"image": PIL 이미지 객체**와 원본 bytes` 데이터**를 담고 있어, 이 역시 응답 직렬화 시 큰 부담이 되었습니다. 이러한 큰 데이터를 Claude Code가 대화 내에 포함시키려다 보니, 25k 토큰 제한을 초과하여 오류가 난 것입니다.

요약하면: **이미지 바이너리 데이터를 응답에 직접 포함**시키는 것이 문제입니다. (NanoBanana MCP 서버는 설계상 이미지를 base64로 반환하도록 되어 있음 [1], 이로 인해 대용량 응답이 발생함.)

해결 방안 (Solution Overview)

해결책은 응답으로 전송되는 데이터의 크기를 줄이는 것입니다. 구체적으로 다음과 같은 조치를 취해야 합니다:

- **Base64 이미지 데이터 제거:** 응답 JSON에서 base64 인코딩된 이미지 데이터를 완전히 제외합니다. 이미지는 **파일로 저장**하고, 필요하면 파일 경로나 파일 크기 정보만 반환합니다.
- **바이너리 바이트 데이터 제거:** `bytes` 형식의 이미지 데이터도 응답에 포함하지 않습니다.
- **PIL 이미지 객체 제거:** PIL Image 객체 자체를 응답 구조에 넣지 않도록 합니다. 객체를 JSON 직렬화하려 할 때 내부 데이터로 인해 크기가 커질 수 있으므로, **이미지 객체는 내부 처리에만 사용하고 결과로는 제외**합니다.
- **대신 참조 정보 제공:** 클라이언트(Claude Code)에는 이미지의 경로나 파일 이름, (원한다면 파일 크기나 메타데이터)만 제공하고, 실제 이미지는 별도로 표시하거나 필요 시 참조하도록 합니다. (예: Claude Code에서는 `【...†embed_image】` 형식으로 이미지를 표시할 수 있으므로, base64 데이터 대신 경로를 이용하도록 유도할 수 있습니다.)

이러한 수정으로 **MCP 응답의 크기**를 크게 줄여 25,000 토큰 한도 이내로 들어오게 할 것입니다. 필요하다면 **ImgBB와 같은 이미지 호스팅을 활용**하여 이미지를 업로드한 뒤 URL을 반환하는 방법도 있습니다. (NanoBanana MCP는 ImgBB API 키를 설정하면 이미지를 업로드하고 URL을 제공하도록 설계되어 있습니다. 이렇게 하면 대용량의 base64를 주고받지 않고 짧은 URL만 전달하게 되어 효율적입니다.)

상세 수정 사항 (Step-by-Step Fix Implementation)

아래는 Nanobanana MCP 서버 코드를 수정하는 구체적인 단계입니다. 각 파일별로 변경해야 할 부분을 정리했습니다:

1. `generate_image.py` 수정

문제점: `image_data["base64"]`를 응답에 포함시켜 거대한 문자열을 반환하고 있었습니다.

조치: `return`되는 딕셔너리에서 `image_data` 관련 필드를 제거했습니다. 대신 이미지 파일 경로와 파일 크기 등만 돌려주도록 변경합니다. 사용자가 이미 이 부분을 다음과 같이 수정한 것으로 보입니다:

```
# (생략)
return {
    "success": True,
    "description": result["text"].strip() if result["text"] else f"Generated image: {prompt}",
    "image_path": str(saved_path.absolute()),
    "file_size_mb": round(saved_path.stat().st_size / (1024 * 1024), 2) if saved_path.exists()
else 0,
    "metadata": metadata,
    "prompt_translation": {
        "original": prompt,
        "optimized": optimized_prompt,
        "auto_translated": Config.AUTO_TRANSLATE
    }
}
```

위와 같이 `image_data`나 base64 필드는 빼고, `file_size_mb` 같은 경량 정보로 대체했습니다. **이 수정은 잘 적용되어야 하며**, 응답에 base64 문자열이 더 이상 포함되지 않게 됩니다.

2. `edit_image.py` 수정

문제점: `generate_image.py`와 유사하게 편집 결과 응답에 base64로 인코딩된 `image_data`를 포함하여 과다한 데이터를 반환하고 있었습니다.

조치: `generate_image.py`와 마찬가지로 `return` 딕셔너리에서 `image_data` 필드를 제거합니다. 이미 사용자가 해당 부분을 아래처럼 수정한 것으로 확인됩니다:

```
# (생략)
return {
    "success": True,
    "description": result["text"].strip() if result["text"] else f"Edited image: {instruction}",
    "image_path": str(saved_path.absolute()),
    "original_path": str(input_path.absolute()),
    "file_size_mb": round(saved_path.stat().st_size / (1024 * 1024), 2) if saved_path.exists()
else 0,
    "metadata": metadata,
    "instruction_processing": {
```

```
            "original": instruction,
            "optimized": optimized_instruction,
            "style_applied": style,
            "auto_translated": Config.AUTO_TRANSLATE
        }
    }
```

위 코드에서도 base64 데이터나 `image_data` 필드를 제거하고, 대신 파일 경로와 크기 정도만 반환하도록 했습니다.

3. `gemini_client.py` 수정

문제점: 이 부분이 **현재까지 남은 핵심 문제**로 보입니다. `gemini_client.py`에서 Gemini API 호출 후 결과를 구성할 때, `result["images"]` 리스트의 각 항목에 다음과 같은 필드들이 있었습니다: - `"image": image` – PIL 이미지 객체 (메모리에 원본 이미지 데이터를 가지고 있음) - `"bytes": image_bytes` – 이미지의 바이너리 바이트 데이터 - `"base64": ...` – (이미 제거했지만 처음에는 base64 문자열)

사용자가 이미 `"base64"` 필드는 제거했고 (`bytes`도 2차 수정에서 제거 시도), 현재 `"image"` 필드가 그대로 남아 있는 상태입니다. **PIL.Image 객체**는 JSON으로 바로 변환되진 않지만, Claude Code가 이 객체를 처리하려고 하면서 내부 내용을 문자열화하거나, 아니면 크기만으로도 제한에 걸릴 가능성이 있습니다. 실제 로그를 보면 base64를 빼도 여전히 180만 토큰 이상의 응답이 발생했는데, 이는 아마 `"image"` 객체가 포함된 `result`를 Claude가 직렬화하면서 생긴 문제일 가능성이 높습니다.

조치: `result["images"]`에 **이미지 객체나 원본 바이트 데이터가 절대 포함되지 않도록** 수정해야 합니다. 이미 이미지를 파일로 저장했기 때문에, 이미지 객체를 응답으로 줄 필요는 없습니다. 따라서 `gemini_client.py`에서 결과 리스트에 추가하는 딕셔너리를 아래처럼 변경합니다:

```
- result["images"].append({
-     "image": image,  # Keep for local processing
-     "format": image.format or "PNG",
-     "size": image.size,
-     "saved_path": str(saved_path)
-     # Removed bytes and base64 to prevent MCP response size issues
- })
+ result["images"].append({
+     # Don't include the image object or raw bytes in the result to avoid large responses
+     "format": image.format or "PNG",
+     "size": image.size,
+     "saved_path": str(saved_path)
+ })
```

위 수정에서는 `"image"` 키 자체를 결과에서 제거했습니다 (`bytes`와 `base64`도 이미 제거). 이렇게 하면 `result["images"]` 리스트의 각 항목은 **경량의 메타정보만** 가지게 됩니다. (`format`, `size`, `saved_path` 정도만 포함)

또한 이 부분 수정 전에, `gemini_client.py` 상단에 `from datetime import datetime` 임포트를 추가한 것도 적절합니다 (이미 하셨을 것으로 보임). 이미지를 저장할 때 파일명에 timestamp를 쓰기 위해 필요했습니다.

Tip: 혹시 Claude Code 상의 `desktop-commander`를 이용해 수정한다면, 다음과 같은 식으로 할 수 있습니다.

- `"image": image,` 부분을 통째로 제거:

```
desktop-commander - edit_block (MCP)(
    file_path: "D:\\nanobananamcp\\nanobanana-mcp\\src\
\gemini_client.py",
    old_string: "\"image\": image,  # Keep for local processing\n",
    new_string: ""
)
```

(참고: 위 명령은 `\"image\": image,  # Keep for local processing` 라인을 찾아 빈 문자열로 치환합니다. 수정 후 쉼표 위치나 줄바꿈이 올바른지 확인하세요.)

- `"bytes": image_bytes,` 부분도 혹시 남아 있다면 제거:

```
desktop-commander - edit_block (MCP)(
    file_path: "D:\\nanobananamcp\\nanobanana-mcp\\src\
\gemini_client.py",
    old_string: "\"bytes\": image_bytes,",
    new_string: ""
)
```

이미 base64 부분은 주석 처리하셨으므로 생략합니다.

4. 서버 재시작

코드 변경을 모두 적용했다면, **MCP 서버를 재시작**합니다. (`src/server.py`를 다시 실행하거나, Claude Code에서 Bash 명령으로 재시작했던 것처럼 진행). 서버를 재기동하여 수정된 코드가 반영되도록 합니다.

5. 수정 후 테스트

이제 동일한 이미지 생성 명령을 다시 시도합니다:

예를 들어:

```
nanobanana - nanobanana_generate (MCP)(
    prompt: "A cute puppy crossing a river, walking through shallow water",
    style: "photo", quality: "high"
)
```

수정이 제대로 되었다면, **더 이상 25,000 토큰 초과 오류가 발생하지 않을 것**입니다. 대신 `"success": True`와 함께 `image_path`, `file_size_mb` 등의 정보가 담긴 JSON 응답이 돌아올 것입니다. Claude Code 상에서는 base64 데이터 대신 비교적 작은 크기의 응답만 처리하므로 안전합니다.

응답 예시 (개략적으로):


```
{
  "success": true,
  "description": "Generated image: A cute puppy crossing a river, ...",
  "image_path": "D:\\nanobananamcp\\public\\generated\
\temp_generated_20250905_105700.png",
  "file_size_mb": 1.45,
  "metadata": { ... },
  "prompt_translation": { ... }
}
```


위와 같이 나오고, **에러 없이** Claude Code가 응답을 받아 후속 처리를 할 수 있을 것입니다.

이제 Claude Code에서 해당 `image_path`를 이용해 이미지를 확인하거나 표시할 수 있습니다. 예를 들어 Claude Code에게 `이미지 파일을 열어서 표시해줘`라고 하거나, 경로를 지정하여 `【...†embed_image】` 형태로 이미지를 출력하도록 할 수 있습니다. (만약 Claude Code가 자동으로 `image_path`를 감지해서 이미지를 보여주는 기능이 있다면, base64 없이도 이미지를 볼 수 있을 것입니다.)

추가 팁 (Optional Enhancements)

- **ImgBB 사용:** 만약 **ImgBB API 키**를 설정해두면, Nanobanana MCP 서버는 생성된 이미지를 자동으로 ImgBB에 업로드하고 응답에 **짧은 URL**을 포함시킵니다. 이렇게 하면 애초에 Claude에게 대용량 데이터를 보낼 필요 없이 URL로 공유할 수 있어 효율적입니다. (단, ImgBB 업로드를 사용하면 이미지가 온라인에 공개되므로 보안/개인정보 이슈는 고려해야 합니다.)
- **품질 조절:** 이미지 크기가 너무 커지는 것이 걱정된다면 `quality: "medium"` 등의 옵션으로 생성하거나 해상도를 제한할 수 있습니다. 하지만 근본적으로는 위 수정처럼 **대용량 데이터 전송을 막는 것**이 가장 중요합니다.

以上의 수정으로 **응답 크기 초과 오류는 해결**될 것입니다. 이제 Nanobanana MCP가 정상적으로 동작하여, Claude Code를 통해 원하는 이미지를 생성하고 확인할 수 있을 것입니다. 문제 해결 후에도 비슷한 대용량 데이터 반환 로직이 없는지 점검하면 좋습니다. 필요한 경우 Claude Code 환경에서 응답을 받아 **토큰 수 추이를 확인**하며 조정할 수 있습니다.

---

1 MCP Nano Banana | Glama
https://glama.ai/mcp/servers/@GuilhermeAumo/mcp-nano-banana