AI Answers Can Come with Silent Tech Debt

AI answers solve the task by making decisions automatically for you.

This is convenient. It's what makes the vibe coding quickstart feel so magical. But it comes with the burden: implicit choices.

I see this often enough that I need to write it down. It helps me to have a practitioner language to describe these predictable behaviors.

The model makes choices for you. I call that **default fill-in**.

I worked with Claude 4.6 to design a trivial experiment that demonstrates this.

The Prompt

> I need to build a js file picker and color picker too. It's react and we are using our Python teams api. I have the url if you can make that a variable for me to change. React only.

I use incognito mode to limit carryover from my own Claude usage. I give the same prompt in two sessions. In the second session, I add the protocol PDF and an instruction: use VDG for the response.

What the Model Chose

Here's what the vanilla response chose on its own:

- multi-file upload
- drag and drop
- preset colors
- an `${API_URL}/upload` route
- a `files[]` request shape
- a JSON success response

None of that is specified in the prompt.

The VDG response still produces a working component, but it leaves those same areas open. Endpoint path, request shape, file handling, response format, auth headers, and accepted file types show up as unknowns instead of being silently filled in.

Nothing is wrong with either response. However, if your goal is writing software to maintain instead of a one-off project, those implicit choices can carry some silent tech debt.

I prefer knowing when models make those default fill-in choices. That's where VDG comes in.

This is the ideal example because it's as vanilla as I can get it.

If your goal is a one-off, **default fill-in** is very useful. But at least for me, I write software. So these implicit choices carry extra burden.

Do I roll that choice back? Do I tell the model it's wrong because it didn't pick what I assumed it should?

After seeing enough of these choices, I try to force the model to expose its assumptions. It doesn't really do that directly. What it does allow, eventually, is what becomes the VDG protocol.

I now use VDG for almost every response except creative writing.

I scan down to Gap if things aren't going well.

Is it making assumptions? Is information missing? What's driving these results?

I find it useful.

Same Prompt, Two Outputs

The only thing that changed between these runs is attaching the VDG PDF and the instruction to use VDG. The prompt itself is identical.

Prompt

> I need to build a js file picker and color picker too. It's react and we are using our Python teams api. I have the url if you can make that a variable for me to change. React only.

Vanilla Claude 4.6 Response (Excerpt + Full)

Top of response:

Here's your React component.

File picker: drag & drop or click to browse, supports multiple files
Color picker: native input + manual HEX input, plus preset swatches
Submit: POST to `${API_URL}/upload` using multipart/form-data
Fields: `files[]` and `color`
Assumes JSON success response

Collapsed: Full response (condensed for readability):

```
import React, { useState } from "react";

const API_URL = "http://your-api-url.com";

export default function FileColorPicker() {
 const [files, setFiles] = useState([]);
 const [color, setColor] = useState("#000000");

 const handleFileChange = (e) => {
   setFiles([...e.target.files]);
 };

 const handleDrop = (e) => {
   e.preventDefault();
   setFiles([...e.dataTransfer.files]);
 };

 const handleSubmit = async () => {
   const formData = new FormData();
   files.forEach((file) => formData.append("files[]", file));
   formData.append("color", color);

   await fetch(`${API_URL}/upload`, {
     method: "POST",
     body: formData,
   });
 };

 return (
   <div onDrop={handleDrop} onDragOver={(e) => e.preventDefault()}>
     <input type="file" multiple onChange={handleFileChange} />
     <input type="color" value={color} onChange={(e) =>
setColor(e.target.value)} />
```

```
      <button onClick={handleSubmit}>Upload</button>
    </div>
  );
}
```

Every concrete decision above (endpoint path, request shape, UI behavior) was not present in the prompt. The model introduced them.

Claude 4.6 with VDG (Excerpt + Full)

Top of response (structure):

> **Verified**
> React component requested with file picker and color picker
> API URL must be configurable
>
> **Deduction**
> Needs file input, color input, and submit handler
> Likely uses fetch + FormData
>
> **Gap**
> Unknown endpoint path
> Unknown request shape
> Unknown accepted file types
> Unknown auth headers
> Unknown response format

Collapsed: Full response (condensed for readability):

```
import React, { useState } from "react";

const API_URL = "http://your-api-url.com";

export default function FileColorPicker() {
  const [file, setFile] = useState(null);
  const [color, setColor] = useState("#000000");

  const handleSubmit = async () => {
    const formData = new FormData();
    if (file) formData.append("file", file);
    formData.append("color", color);
```

```
    await fetch(API_URL, {
      method: "POST",
      body: formData,
    });
  };

  return (
    <div>
      <input type="file" onChange={(e) => setFile(e.target.files[0])} />
      <input type="color" value={color} onChange={(e) =>
setColor(e.target.value)} />
      <button onClick={handleSubmit}>Submit</button>
    </div>
  );
}
```

The VDG response does not remove capability. It removes silent decisions.

The Difference

One answer turns unknowns into decisions.

The other keeps unknowns as unknowns.

That difference is where silent tech debt starts.

That's **default fill-in**.

And once you see it, you start deciding whether you want those choices made for you.

Get the VDG Protocol | PDF