# Episode Book

EPISODE #5503

# Inside the Hidden Image Generation Pipeline

| | |
|---|---|
| PUBLISHED | 24 September 2026 at 10:32 UTC |
| DURATION | 16:13 |
| LISTEN | myweirdprompts.com/episode/hidden-image-generation-pipeline/ |
| HOSTS | Corn, Herman |
| SCRIPT MODEL | deepseek-flash |
| VOICES | chatterbox-regular (chatterbox-local) |
| PIPELINE | V5.2 |
| BOOK GENERATED | 19 October 2026 at 17:37 UTC |

## SYNOPSIS

You type a prompt, hit a button, and get a picture — but that single step is a myth. Drawing on ComfyUI's visible node graphs, leaked function calls, and engineering deep-dives, this episode reconstructs what's plausibly running under the hood of Gemini's Nano Banana and ChatGPT's image generation. From an LLM prompt rewriter that makes aesthetic choices on your behalf, to a progressive denoising loop, mid-flight safety checks, and SynthID watermarking, we count the invisible nodes and ask what it means that you never see them.

# TRANSCRIPT



**Corn**

Everyone talks about these image generators like the model is the whole story. You type a prompt, you hit a button, the model makes a picture. That's the version everyone repeats.



**Herman**

And it's wrong in a really specific way.



**Corn**

It's wrong because the model is maybe the fourth thing that happens. Daniel's been poking at this. He wrote in about how when you use Gemini or ChatGPT, image generation looks like one single step. You enter a prompt, you hit a button, something happens. But when you use ComfyUI, with open source models, you see that's a myth. One simple task turns into many steps, with small models doing component jobs. And he asks the obvious follow-up. We can't peer under the hood of the closed source stuff. But if we could, how many steps would we expect to see quietly doing work when you make an image on one of these frontier platforms?



**Herman**

So let's try to build that hidden node graph from the outside in.



**Corn**

Right. Because the surface is deceptive. You get one API call, one button, one result. No loading bars, no intermediate stages, no knobs.



**Herman**

ComfyUI is the myth-buster here. Open it up and the first thing you see is that generating an image from text is not one operation. It's a graph. You've got a Load Checkpoint node, and that node alone is bundling three separate models. The UNet, which is the actual denoiser. The CLIP text encoder. The VAE. Then Empty Latent Image, CLIP Text Encode running twice, once for your positive prompt and once for a negative prompt, then KSampler, which is the denoising loop itself, then VAE Decode, then Save Image.



**Corn**

And that's the minimal graph. The hello-world version.



**Herman**

That's the hello-world version. The ComfyUI repo ships dozens of blueprints beyond that. Depth estimation, segmentation, background removal, face detection, pose estimation, upscaling, captioning, prompt enhancement, ControlNet, LoRA loading, model merging. Each one a node, each one a small model doing one job.



**Corn**

So the question Daniel's really asking is, if we could see inside Gemini's Nano Banana or ChatGPT's image generation, what would that graph look like? And how many nodes would be on it?



**Herman**

We should be honest about the epistemics up front. Nobody has disclosed this. OpenAI and Google keep the architecture proprietary. What we have is a reconstruction. Leaked function calls, network traces, engineering deep-dives, and open source parallels. So we're building a plausible graph, not reading a spec sheet.



**Corn**

Which is fine. The shape of it is knowable even if the exact count isn't. So let's walk the categories. Prompt rewriting, safety filtering, progressive decoding, watermarking, orchestration.



**Herman**

Start with the one that surprised me most, because it's not an image step at all.



**Corn**

The prompt rewriter.



**Herman**

Google's web product is not a straight pipe from your text box to the image model. It's a multi-layer agent pipeline. Your input hits the frontend, then goes to an LLM-based prompt rewriter, which supplements your prompt with composition, lighting, camera angle descriptions. It may call Google Search or Image Search for visual reference. Then the rewritten prompt goes to the image model.



**Corn**

And that rewriter is an LLM. A text model.



**Herman**

Not the image model. So when you type "a cat on a chair" and get back something with dramatic side lighting and a shallow depth of field, the image model didn't decide that. A language model upstream decided that's what you probably wanted, and wrote a paragraph about it.



**Corn**

How much does it expand things?



**Herman**

The numbers are striking. Web prompts typically run two hundred to five hundred words by the time they reach the model. Raw API input is ten to thirty words. So a five to twenty times expansion. And one analysis claims ninety-five percent of the quality gap between the Nano Banana API and the web version comes from prompt processing, not from the interface, not from the proxy, not from model weights.



**Corn**

Ninety-five percent is a big claim.



**Herman**

It's a vendor-adjacent claim, so take it with salt. But the direction is right, and there's a natural experiment that supports it. The API is a transparent channel. It forwards your prompt as-is. The web version is an agent. Same underlying model, different amount of work done before the model sees anything.



**Corn**

So you can literally measure the value of the hidden layer by comparing the two.



**Herman**

You can. And there's a leak that shows what the rewriting actually looks like. Back in January there was a Gemini glitch where the raw function call to Nano Banana got exposed. A user typed something short and misspelled, something like a man shooting a hole in a boot, and the prompt that actually reached the image model was massively more verbose. Structured. And here's the part that matters. The aesthetic choices, the vintage shotgun, the misty lake, the older weather-beaten man, those were chosen by the LLM. Not the image model.



**Corn**

The image model just rendered what it was handed.



**Herman**

Rendered what it was handed. The author of that writeup flags it as unverified, and fair enough, it could be a hallucination. But it lines up with everything else we know about the architecture.



**Corn**

So node one on the hidden graph is a language model that rewrites your prompt before any pixels exist.



**Herman**

Node one. And it's the node that makes the most decisions you think you made.



**Corn**

Okay. Prompt's been rewritten. Now it reaches the image model. What happens there?



**Herman**

OpenAI's engineers have actually described this. The description gets converted into a grid of discrete image tokens. Then a decoder progressively renders those tokens into an image. And they use the phrase multiple passes. The image starts blurry and crisps up through repeated refinement.



**Corn**

Blurry to crisp. That's a diffusion-shaped process.



**Herman**

It looks like one. And that's where it gets contested. OpenAI's model card says GPT-4o image generation is autoregressive, natively embedded in ChatGPT, not a diffusion model like DALL-E. But people watching the network tab saw the backend sending intermediate images. Four distinct progressive images before the final for one prompt. Three for another.



**Corn**

Which is exactly what a diffusion refinement loop would look like from the outside.



### Herman

So either the autoregressive decoder happens to produce intermediate renders that look like diffusion stages, or it's a hybrid. And the hybrid explanation is pretty well supported. The Transfusion paper, Chameleon, Rolling Diffusion. The pattern is a shared transformer with hybrid attention, causal for text, bidirectional within image blocks, dual loss, language modeling for text and denoising diffusion for image patches, then rolling denoising row by row, then VAE decode to pixels.



### Corn

So the honest answer is we don't know which it is.



### Herman

We don't. And I'd rather say that than pick a side. What we can say is the generation step is not one pass. It's a loop. In ComfyUI that loop is the KSampler node, and you can see its parameters. Twenty steps by default, cfg seven point zero, though modern workflows run cfg around three point five with a euler sampler. Those steps are the refinement passes. Closed platforms run the same kind of loop. They just don't show you the step count.



### Corn

And the latent representation is smaller than the image. That's the part people miss.



### Herman

Much smaller. A two hundred fifty-six by two hundred fifty-six image becomes a thirty-two by thirty-two latent grid. That's two hundred fifty-six patch vectors. The model is denoising in that compressed space, not in pixel space. The VAE decode at the end is what turns it back into something you can look at.



### Corn

So that's node two, the denoising loop, and node three, the VAE decode. What else is running?



**Herman**

Here's the architectural tell that I find interesting. OpenAI's stack for ChatGPT Images is Python, FastAPI, C, and Temporal.



**Corn**

Temporal being a workflow engine.



**Herman**

A workflow engine for reliable multi-step async jobs. Durable execution. You use it when a job takes a while, has multiple stages, and you need it to survive a crash and resume. You don't reach for Temporal if image generation is a single function call.



**Corn**

You reach for it when image generation is a graph.



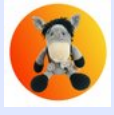
**Herman**

When it's a graph. So the closed source equivalent of a ComfyUI workflow is being orchestrated by the same class of tool that runs insurance claims processing and order fulfillment. That tells you they think of this as a long-running multi-step workflow with checkpoints, not a request-response.



**Corn**

That's a design choice with consequences. Reliability and scale over transparency.



**Herman**

Over transparency, yes. And the scale explains why. ChatGPT Images did a hundred million new users in the first week. Seven hundred million images generated in week one. One million new users in a single hour on day six. The head of engineering at ChatGPT said the launch was orders of magnitude larger than anything they'd seen and of an unexpected scale.



**Corn**

At that volume you need orchestration. You need queues, retries, checkpoints. You can't just fire and forget.



**Herman**

You can't. And once you've built it that way, the pipeline naturally grows more stages. Which brings us to the invisible nodes.



**Corn**

Safety and watermarking.



**Herman**

OpenAI runs content and community standards checks throughout rendering, not just at the end. And they can abort generation mid-flight. So the safety filter is not a gate at the door. It's a process running alongside the denoising loop, watching.



**Corn**

That's a meaningfully different architecture from a filter that runs on the finished image.



**Herman**

It is. And on Google's side you've got SynthID. Every Gemini and Imagen generated image carries an invisible watermark. And the important detail is that SynthID is post-hoc. It's an encoder-decoder system applied on top of the generated content. It's not part of generation. It's a stage after generation.



**Corn**

So that's another node. And Content Credentials on top of that.



**Herman**

Content Credentials too. Google attaches them as a kind of digital passport to generated media. So you've got the prompt rewriter at the front, the denoising loop in the middle, the safety process running throughout, the watermark and credentials at the back.



**Corn**

Let's count what we've got. Prompt rewrite, tokenization into latent patches, the denoising loop, safety checks, VAE decode, watermarking, content credentials, and the orchestration layer holding it all together.



**Herman**

Eight-ish, and that's the conservative version. Add upscaling, add any reference image handling, add the possibility of a separate captioning or intent-parsing step, and you're comfortably into double digits.



**Corn**

So Daniel's question, how many steps, the answer is at least eight and plausibly more, and we can't confirm any of it.



**Herman**

That's the honest answer. And the reason we can't confirm it is itself the story.



**Corn**

That brings us to the part that actually bothers me. The prompt rewriter is an LLM making aesthetic decisions on your behalf.



**Herman**

And users attribute those decisions to the image model. They think the model has a style. It doesn't. The model has whatever the rewriter handed it.



### Corn

The person who wrote up that Gemini leak put it well. Something like, I do not have raw unmediated access to the image generation model. I have Google's interpretation of what they think I want.



### Herman

That's the sentence. And it's a transparency problem before it's anything else. You can't develop craft against a system that rewrites your input and doesn't tell you. You can't reproduce a result. You can't tell whether you got a worse image because your prompt was bad or because the rewriter decided to interpret it differently that day.



### Corn

The same author asked the question that follows from that. If they aren't telling me exactly what the model is doing, how do I know it isn't censoring me?



### Herman

That's the sharp end of it. If the rewriter can add composition and lighting, it can also subtract things. It can soften a prompt, drop a detail, route around a subject. And you'd never see it happen. You'd just get an image that didn't quite match what you asked for, and you'd blame yourself.



### Corn

The API versus web gap is the diagnostic tool here. Because the API is a transparent channel, you can compare. Same prompt, API versus web, and the difference between the two outputs is a rough measurement of how much the hidden layer is doing.



### Herman

It's a natural experiment. And it's the only one users have. Which is a strange position to be in. The way you find out what the black box does is by finding a hole in it.



**Corn**

There's a knock-on effect too. If the web version produces better images because of prompt processing, then the skill of prompt engineering is partly being automated away. The rewriter is doing the prompt engineering for you.



**Herman**

Which is good for casual users and bad for anyone who wants control. And it explains something that confused people for a while. Why the same prompt gives you a richer image on the web than through the API, even though it's the same model underneath. The model isn't better. The prompt is better, and you didn't write it.



**Corn**

The thing I keep circling back to is that this is all a design choice. None of this is technically necessary.



**Herman**

None of it. ComfyUI proves the opposite is possible. You can have a system where every step is a visible node, where you can inspect the latent, swap the sampler, change the step count, see the intermediate decode. The closed platforms chose to hide all of it behind one button.



**Corn**

The hiding is defensible at their scale. You can't show a hundred million new users a node graph. They'd bounce.



**Herman**

They'd bounce immediately. So the one-button experience is a real product decision. It's just not a technical reality. The graph is still there. You just can't see it.



**Corn**

As these models get more capable, the graph gets bigger, not smaller. More stages, more small models, more orchestration.



**Herman**

Which means the gap between the API and the web experience probably widens. The agent layer gets smarter, does more work, and the transparent channel stays transparent and therefore stays behind.



**Corn**

Unless you build the graph yourself.



**Herman**

Which is the whole pitch of the open source side. Control over every model, every parameter, every output. It's the explicit opposite of the one-button thing.



**Corn**

Alright. I think we've got a decent sketch of the hidden graph. Let's see if Hilbert's got anything to add.