Stable Diffusion Doesn't "Know" Anything

Paul R. Potts

December 2022

I'm not an expert on artificial intelligence, but I'm going to take a crack at explaining and discussing a recent innovation, and why it is, under the hood, both more and less impressive than it seems at first glance.

Background: Old AI

In the early years of artificial intelligence, there were many attempts to make it so that computers could "know" things, and demonstrate that they could reason about the things that they "know." (I'm putting the word "know" in scare quotes because I think there's a legitimate philosophical question to discuss, as to whether computers can know anything the way humans know things, and of course whether humans can ever know things, but that is far beyond the scope of what I want to talk about just now.)

Anyway, one of the most famous examples of an artificial intelligence that could "know" things is called SHRDLU:

SHRDLU

> SHRDLU was an early natural-language understanding computer program, developed by Terry Winograd at MIT in 1968–1970. In the program, the user carries on a conversation with a computer, moving objects, naming collections and querying the state of a simplified "blocks world," essentially a virtual box filled with different blocks.

Here's part of a transcript, taken from Terry Winograd's web page:

```
Person: HAD YOU TOUCHED ANY PYRAMID BEFORE YOU PUT THE GREEN ONE ON THE LITTLE CUBE?
Computer: YES, THE GREEN ONE.
Person: WHEN DID YOU PICK IT UP?
Computer: WHILE I WAS STACKING UP THE RED CUBE, A LARGE RED BLOCK AND A LARGE GREEN CUBE.
Person: WHY?
Computer: TO GET RID OF IT.
Person: WHY DID YOU DO THAT?
Computer: TO CLEAN OFF THE RED CUBE.
Person: WHY DID YOU CLEAR OFF THAT CUBE?
```

```
Computer: TO PUT IT ON A LARGE GREEN CUBE.
Person: WHY DID YOU DO THAT?
Computer: TO STACK UP THE RED CUBE , A LARGE RED BLOCK AND A LARGE GREEN CUBE.
Person: WHY DID YOU DO THAT?
Computer: BECAUSE YOU ASKED ME TO.
Person: HOW DID YOU DO IT?
Computer: BY PUTTING A LARGE RED BLOCK ON THE TABLE ; THEN LETTING GO OF IT ; THEN PUTTING A
```

This transcript illustrates that not only does SHRDLU "know" things about the objects in its virtual world, but it also maintains a history of the things it has done in that world; note also that it took some "secondary actions" to facilitate the actions it was asked to do directly.

SHRDLU was featured in one of the books that deeply influenced my early intellectual life, Douglas Hofstadter's *Gödel, Escher, Bach: An Eternal Golden Braid*. I'm not sure if I first read about it in that book; it may have been featured in something else I came across, such as *Scientific American* magazine. SHRDLU was built partly in Lisp and partly in a language called "Micro-Planner," which was a "domain-specific language" or DSL, also implemented in Lisp.

Micro-Planner Micro-Planner was a predecessor of the Prolog programming language. Prolog was, and still is, one of the most popular and most practical languages built to support logic programming. It was practical enough that in the 1990s, my team at the University of Michigan, particularly my former co-worker Alan Grover, experimented with using Prolog to generate tailored text messages based on user input, following an elaborate system of rules, although ultimately we wound up using a combination of Perl and Scheme instead. It isn't as widely used as C++ and Java, but it remains a great example of the logic programming *paradigm*.

Four Paradigms For an overview of four major programming paradigms, I highly recommend this lecture about four exemplary languages from the 1970s. (I'm using the word "exemplary" in its more literal meaning, "serving as an example of a particular school of thought," rather than its more common meaning, "one of the best of.") The four languages, from what the author calls "a golden age of programming languages," are SQL, Prolog, ML, and Smalltalk. These languages represent real implementations of four important programming *paradigms* and so are still very relevant and worthy of study today. In that talk, Scott Wlaschin mentions another paradigm that has become so popular that most of the folks that solve problems with them don't even realize they are doing programming: the spreadsheet.

In fact, I'd go so far as to say that I haven't seen very many *more* major programming paradigms since then. Most of the big modern languages — such as JavaScript, Python, Haskell, C++, etc. — really just improve on, and blend together, those four paradigms. (In practice, languages which have good support for multiple paradigms seem to be more successful, long-term.) There have

been a few other interesting programming paradigms over the years, such as stack-based languages, functional reactive languages, and dataflow languages (a spreadsheet can be considered an implementation of a dataflow language; the other widely-used one is LabVIEW), but I don't consider these to represent all that much recent *theoretical* innovation.

Anyway, back to Lisp, Prolog, and Micro-Planner. Micro-Planner allowed SHRDLU to maintain a set of rules, make inferences using the rules, and create new rules, as guided by user commands. Many of the most famous examples of artificial intelligence programming from that era were built using "inference engines" that allowed computers to "know" things, at least in a limited domain, and were enough to impress the demo audience and make history.

Here's a vintage film, unfortunately silent, showing what SHRDLU could do. Note that handling queries in fairly natural English, at least a limited subset of English, was also quite impressive for the time, although for the purposes of this discussion I want to focus on how the system is able to learn and "know" things about its simple virtual world, now how it can understand English and reply to users in English.

Stable Diffusion

In late 2022, the Internet is abuzz with talk about Stable Diffusion and ChatGPT. I haven't looked into ChatGPT much yet, but Stable Diffusion is pretty amazing. If you want to have your mind blown, take a look at this video, in which every frame was generated by Stable Diffusion.

How it Works When I first read about Stable Diffusion, I found it quite difficult to understand just how it works, since the mathematical model behind it are somewhat beyond me, and I haven't kept up with the theory, but after experimenting with it, it started to make more sense. What follows represents my attempt to explain Stable Diffusion to non-specialists and therefore by definition can't be completely inaccurate, but I think it should cover the basics.

At its heart, *diffusion algorithms* are descendants of algorithms originally designed to reduce noise in images. These kinds of thing has existed for decades; for example, some Photoshop filters can remove noise from images, but these algorithms are more sophisticated, in that they don't just compare pixels in an image against their neighbors and smooth them out, but they apply a *model*, also known as a *map*, that guides their noise-reduction strategy, according to what the image looks like. A diffusion *map* is a multi-dimensional mathematical construct that (somehow; I'm leaving out a lot here) represents the *information* in the image, not just the pixels.

Start by imagining an algorithm that can clean up an existing image by removing noise. Imagine that this is done using a map, a multi-dimensional mathematical construct, which has been "trained" on images, so that when a noisy image comes in, the map can spit out a slightly cleaned-up image. If you like the resulting

image, because it is closer to what you had in mind, you can repeat the process — you can *iterate*, feeding the output of the first step through the process again. That's the core idea behind Stable Diffusion's iterative image-processing.

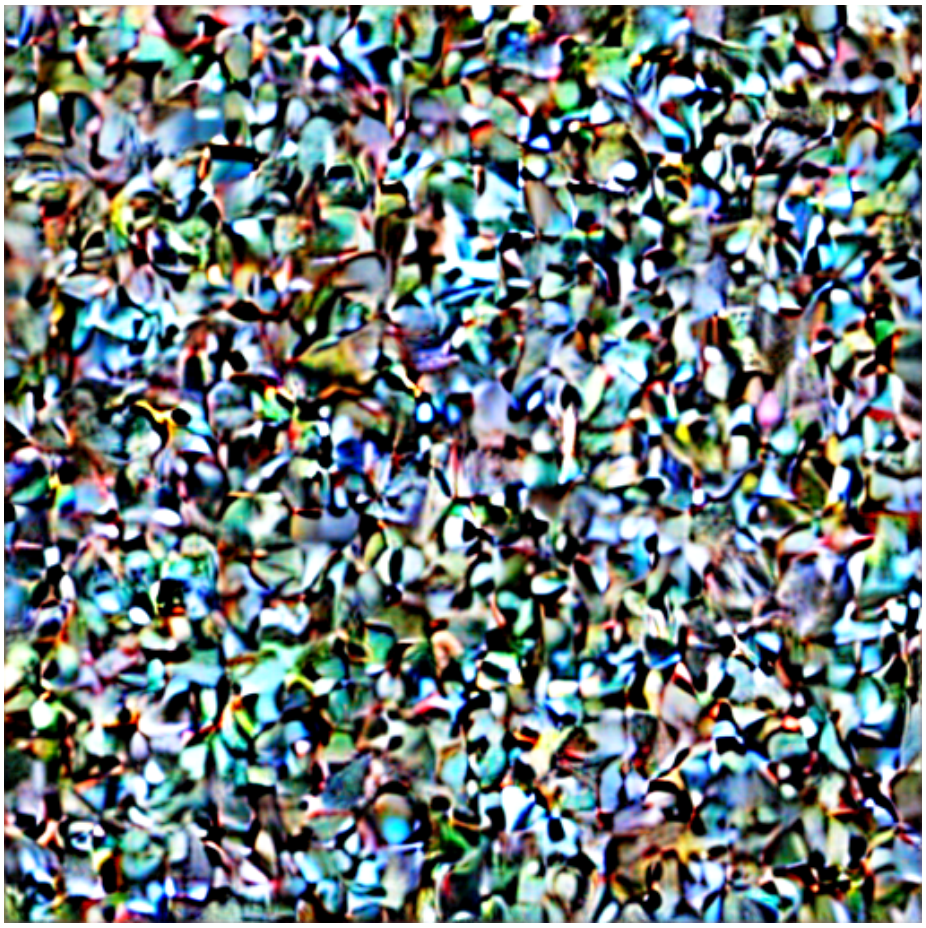
What happens if you start with an image that is purely random noise — that looks like old television static? Well, the algorithm tries to clean it up, and with each iteration it tunes the noise into something that looks more like an intentional image. The results are highly dependent on the exact pattern of noise in the original image, and the way the map has been "trained."

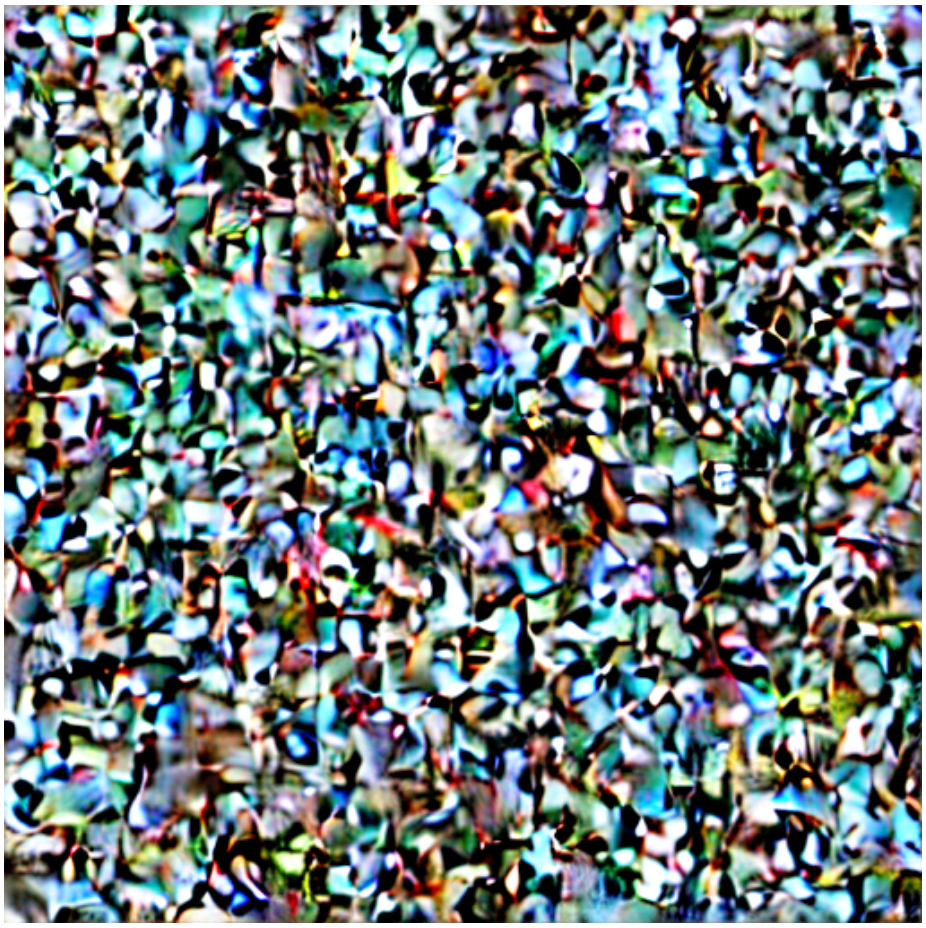
Now imagine that the map has been blown up into an enormously, complex, and quite large data structure. It's been trained on hundreds of millions of existing images. The output has been evaluated and weighted by teams of humans, and it has as many as a *billion* values that can be adjusted: that is, "knobs you can turn" to change the way it works. Turning any of these knobs will slightly alter what might come out of an iteration.

The map isn't "smart" like SHRDLU was. It doesn't "know" anything about the images. It's just a huge cloud of numbers in a multi-dimensional mathematical space, with those "knobs" (parameters) that can adjust how it works. There's no human-readable program in MicroPlanner or any other human-readable language, although of course the tool that *applies* the map is implemented in code. But what it's doing isn't *legible* to a human the way the core logic of SHRDLU is; it makes its decisions based on clouds of numbers.

Since the "knobs" that adjust the behavior of the model aren't labeled, and there are so many of them, how can the user possibly control them? Well, that's where the text prompts come in. There are *more* models that are capable of mapping words in a text prompt to adjustments to the "knobs."

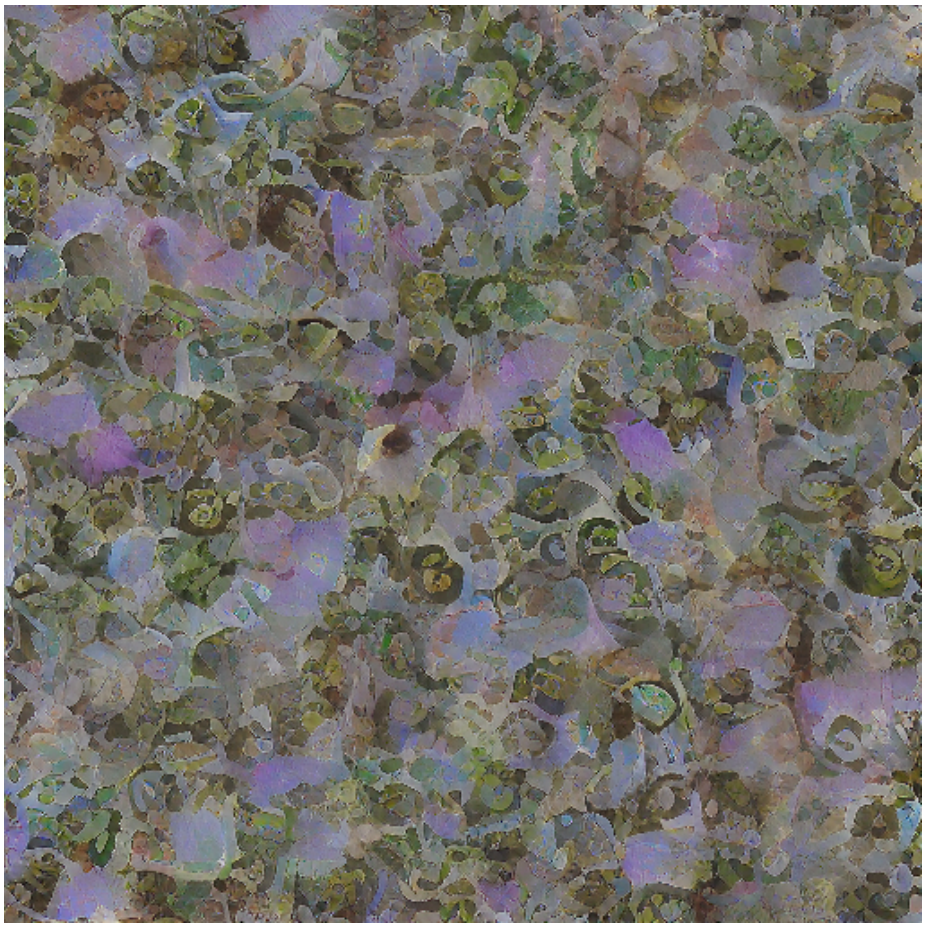
Examples Let's start to make this more concrete: I have a version of Stable Diffusion called "DiffusionBee" running on my MacBook Air laptop. Let's say I give it a text prompt consisting of a single word: "green." Remember, it starts from pure noise. If I tell it to apply the algorithm once only — a single iteration — the generated images look like this:





As you can see, there is some low-level, *localized* structure emerging — blobs of color, a little texture — but not much. The images look like what they are: random noise that's been slightly "de-randomized," steered towards something a bit less random. Note also that there's not very *green*. This almost looks to me like "phosphene noise" you might get when rubbing your eyes, after your visual cortex has run its own algorithms (on real neurons, not neural nets) to try to "de-noise" it into something recognizable.

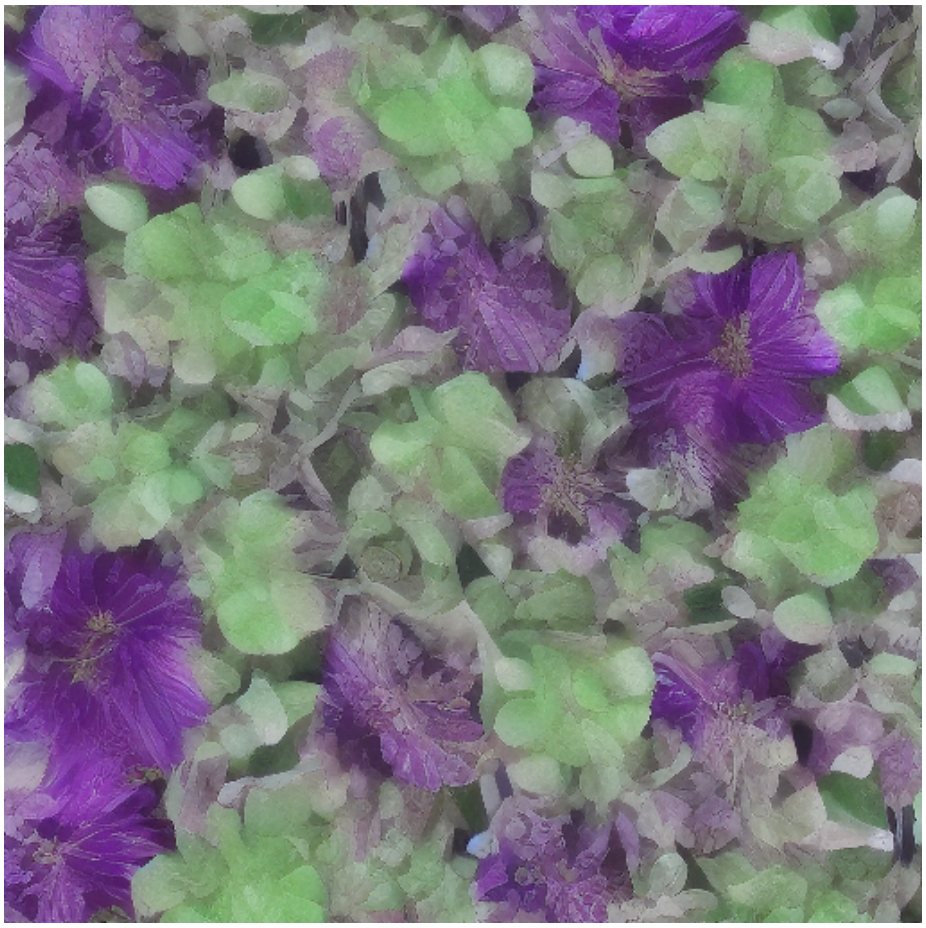
As the algorithm iterates, the images rapidly start to look like they're being steered towards something more recognizable, although where they go is different each time, dependent on the random starting conditions. Here are three from a batch I generated using two iterations:



I don't have an easy way to apply a second iteration to the images generated above, unfortunately, so it's hard to see clearly how they start to diverge from each other, but hopefully this will get the idea across. As you can see, it's still hard to tell where the algorithm is "going." That's because it doesn't really know — remember, all it can really do is to make an image look slightly more like another image, bit by bit, by using a mathematical model that has been trained on millions of pictures, and tuned this time to head towards images to which the word "green" is somehow applicable. So, naturally, these images are looking a little more greenish. These look like they might be heading towards a picture of aquarium gravel, or maybe a succulent plant, or maybe an abstract painting.

If we take 3 steps, we can see more structure emerge, and maybe start to guess at what images the algorithm is pushing the noise *towards*. Maybe towards a watercolor painting, or a field of flowers, or an aerial photograph of the woods somewhere.

By the time we get to four iterations, we can see perhaps a field of lavender plants, or a close-up of a branching herb, or morning glories:



I saved examples of five and six iterations, but they aren't that interesting; they look much like the four-iteration images. However, by the time we get to *seven* iterations, we usually have recognizable images, although keep in mind that they are *generated*; Stable Diffusion doesn't generate an *exact* copy of any of the images in the corpus it was trained on. Here's a road in a fantasy forest that probably looks *somewhat* like several of images that were in the training data; I'm assuming that if the code build was instrumented, and did enough logging, it might be possible to trace the output to the original training images that had the biggest influence. But it's still a bit blurry and dream-like:

Here's another image generated by seven iterations, which went down a very different path:

It's interesting: with just the single prompt word, "green," most of the images that were generated by my experiments were of women in green dresses. Maybe the training data included an over-abundance of pictures of women in green dresses. Here's an image generated by *fifty* iterations:

Figure 1: A realistic road with curbs leading down a hill between tall, very bright green sunlit trees, blurry

Figure 2: A blurry, twisted image of two women wearing different styles of green dresses in front of a brown background

Figure 3: A group of seven women in nearly identical green dresses, with blurry, distorted faces and limbs

The *dressses* mostly look reasonable, at least if you don't look too closely, but the people wearing them can only be described as "pretty f****d up." As I'll discuss, Stable diffusion doesn't know how many legs people have, or how fingers work. And more iterations don't always help; twenty-five seems to be more than enough, in most cases, to get an image that is about as good as it's going to get.

The prompts can help a lot, and a detailed prompt will often result in a much more realistic image. Here's an image of a beetle (sort of) that doesn't exist:

Figure 4: Photorealistic brightly colored image of a bright green, shiny beetle-like insect on a pale blue background

It looks like a cross between a beetle, a spider, and a bee, although it doesn't have the expected number of legs. I generated this image using the prompt "Green beetle on a blue background, realistic photograph, high dynamic range."

And now, here's an image that I quite liked; I made it using the prompt "Elon

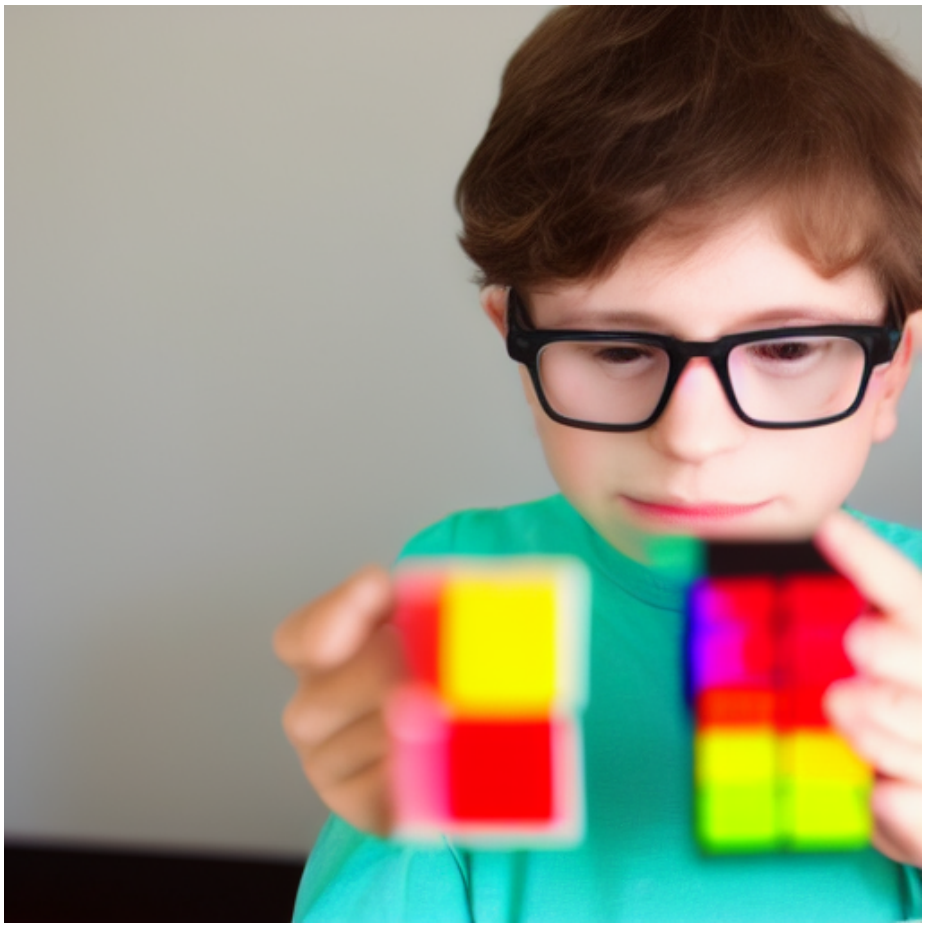
Musk in a giant mech suit waving a giant glowing sword at a huge glowing blue Twitter bird logo, science fiction film scene."

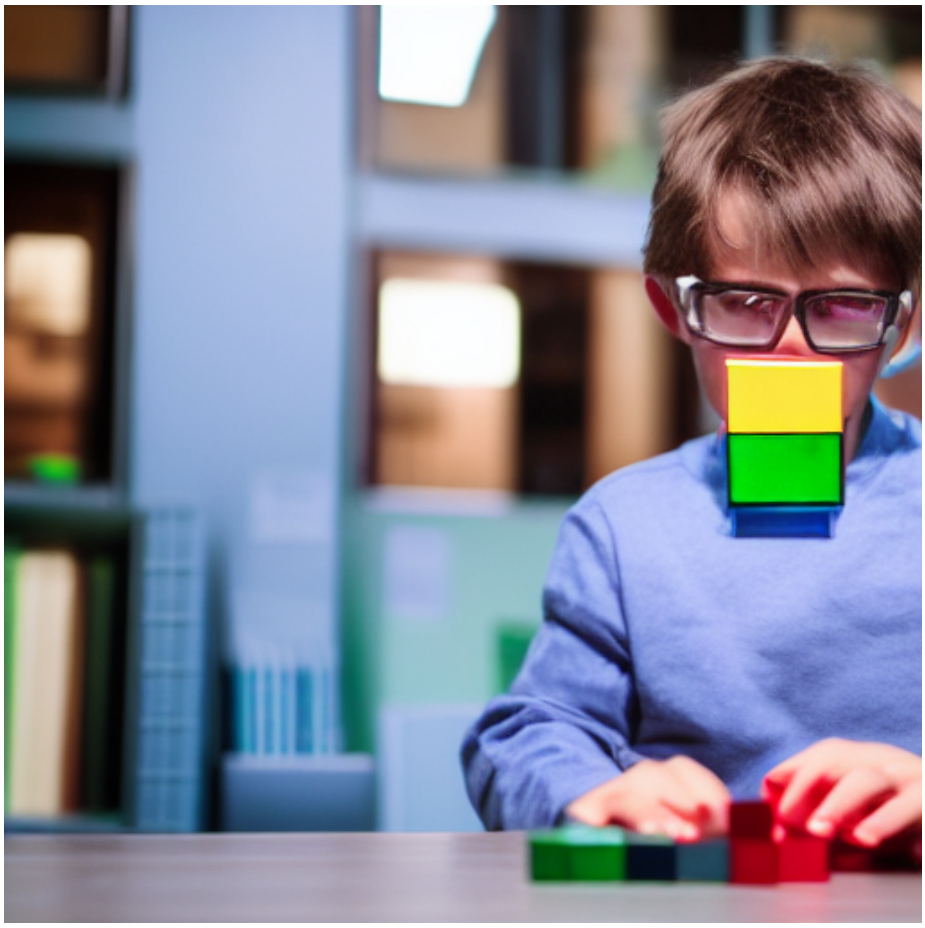
Figure 5: Painting-like image of Elon Musk, brilliantly lit, wearing a complicated spacesuit without a helmet

As you can see, though, he's not in a giant mech suit, and he's not waving a glowing sword, and there's no Twitter bird logo. Because, I mentioned, Stable Diffusion doesn't really *know* anything about the images it's producing. Here are "six pencils lined up neatly in a row," according to Stable Diffusion:

If I ask it to generate a nerdy boy playing with a Rubik's Cube, it will have kind of the right idea:

Figure 6: Closeup picture of three sharpened colored pencils





But it makes *local* changes to the image in each iteration. It doesn't know that the two halves of the cube need to be attached to each other. It doesn't know how fingers work. And you can't ask it to explain how it got there.

The Ethics of Stable Diffusion: No Conclusions, Only Questions The online discussion about Stable Diffusion tends to lean in several directions. A lot of people have written about how this tool is unethical because it uses the work of photographers and graphic designers without their permission.

That's true, but much hinges on the definition of "uses" here. It doesn't directly spit out the images it was trained on, or even portions of them. But what if Stable Diffusion's output is used for commercial purposes — should the creators of all the images that infuenced the model to create the output get compensated, or at least credited? (I'm not sure if that is even possible, without blowing up the model to many times its current size, requiring far more memory; and what can be done if images from a hundred thousand creators demonstrably affected

the image-generation process?)

What responsibility do the creators of these tools have, if the tools are used to create pornography, or other kinds of images deemed harmful? (Yes, Stable Diffusion will create sexually explicit images, and there are no doubt a lot of sexually explicit images in the training data set, although the results will tend to be more surreal and puzzling than arousing; remember that it doesn't "know" how arms, legs, and other body parts work, or even how many of each there should be.)

What happens when real people's faces, or images that look very much like real people's faces, show up? Are their legal, financial, or moral obligations associated with the algorithm that has been trained on the likenesses of many, many recognizable people, and so sometimes spits out recognizable faces, especially, but not only if their names are included in the prompts? (For example, see my image of Elon Musk above; I also got a pictures of people that look vaguely like Billie Joe Armstrong and Billie Eilish accidentally, without actually mentioning their names in the text prompts, while trying to get a picture that looked approximately like Veronica:

Neither of those images seems to be very exact; I can't use the Tin Eye reverse image search to find any matches. I experimented with some services that do more approximate matches, but didn't get any closer to finding a *specific* source image. But is there legal impications when the tool generates a likeness of a real person, even if isn't an exact copy of a real photograph?

What happens when explicit images are combined with real people's faces?

Is it possible to monetize Stable Diffusion — to make it a profitable product? And if not, how is development funded? After all, the computer power and storage required to train models on millions of images, or hundreds of images, can't be cheap, and neither is the process of attaching textual words and phrases to source images.

It's an amazing technology demo, but is it a useful tool for generating images for arbitrary uses, including commercial uses?

Who owns the results?

I think it might eventually make sense when used as an adjunct, or in combination, with different forms of artificial intelligence that *do* "know" things, such as tools with physics models in them. Such "hybrid" tools to me seem immensely promising. I've seen one such example already: a plug-in that allows peope to render 3-D models in Blender, and then create Stable Diffusion-generated images to wrap around their surfaces. The results of that are a bit crude-looking, like the object texture maps used in early video games, but the possibilities are there.

What do you think?

I intended to write more in this newsletter, but this little essay has taken a while to complete, so I'm going to wind up with:

A Festive Gallery of Real and Fake Potts Kids

Real Elanor:

A drawing of a fake Elanor:

Real Malachi:

Fake Malachi:

That one was generated by the text prompt "My son Boobah cryiing because we wouldn't let him eat a spoonful of rasberry jam right out of the jar, Renaissance painting" (based on actual events).

Real Benjamin:

Fake Benjamin:

Real Daniel Peregrine (Pippin):

Fake Pippin (I wound up with many, but this one was interesting):

Note that my text prompt included "candles," but somehow that turned into a single burning candle wick floating in space.

Real Joshua:

I did a number of experiments with this photograph of Joshua, including testing some alternate ways of using Stable Diffusion to create new images by extending existing images, or re-painting parts of existing images, combined with text prompts. I wound up creating some uncanny friends for Joshua; here's one:

Note that it kept the windows, but changed what is seen through them, and it kept the *concept* of "candles," but replaced them with a different version of the concept!

Here's an attempt at extending the scene:

And here is an interesting result I got when I asked Stable Diffusion to repaint part of the original image of Joshua in the window, along with the text prompt

Figure 7: “Real Elanor”

Figure 8: “Fake Elanor”

Figure 9: “Real Malachi”

Figure 10: “Fake Malachi”

Figure 11: “Real Benjamin”

Figure 12: “Fake Benjamin”

Figure 13: “Real Pippin”

Figure 14: “Fake Pippin”

Figure 15: “Real Joshua”

Figure 16: “Joshua’s Uncanny Playmate”

Figure 17: “Joshua’s Extended Image”

describing Joshua and the candles. This picture illustrates how the algorithm can work when you don’t start it off with random noise, but with a portion of an existing image:

I saved this one because while the result was not realistic, it was interesting. Stable Diffusion made a little companion for Joshua, following the text prompt, and even made some more candles out of the blurry wooden peg dolls that were lined up on the window sashes. The white sash, a horizontal surface, seems to have been “tuned” into a place to put candles, probably because in the training images, candles could be often found sitting on a horizontal surface. The peg dolls themselves appear to have served as “seed shapes” that triggered the algorithm, with its “knobs” adjusted by the text prompt, to place candles there, making them progressively, on multiple iterations, look more like candles.

It did this without actually “knowing” anything about the way that candles work; note that the placement of the boy itself doesn’t make any kind of real-world, physical sense; it seems to make a kind of “image logic”. One of the seed images might have contained a boy in front of a curtained window. The vertical lines in the window frame seem to have triggered the text-prompted algorithm to draw a pattern of pixels that it associated with the word “window,” even though the prompt didn’t mention a curtain, and the original image didn’t have curtains in it.

That’s a fascinating little window, so to speak, into the way the algorithm “knows” things and “thinks” things. Is it, very generally speaking, and very broadly, akin to the way our minds, tuned by our experiences, recognize and name images, but run in reverse, “imagining” things that aren’t there?

Here’s the real Sam:

Figure 18: “Joshua’s Candle Friend”

Using a text prompt, I somehow wound up getting a fake Sam that looks remarkably like the real Sam:

The prompt was “16 year old boy with large afro, skinny, wearing a hooded sweatshirt, Renaissance painting.” Clearly, it’s evidence that Sam will one day become a time-traveler.

Real Veronica:

Fake Veronica:

I generated the fake Veronica with the text prompt “18 year old girl with punk haircut and earings, wearing a blanket, charcoal drawing.”

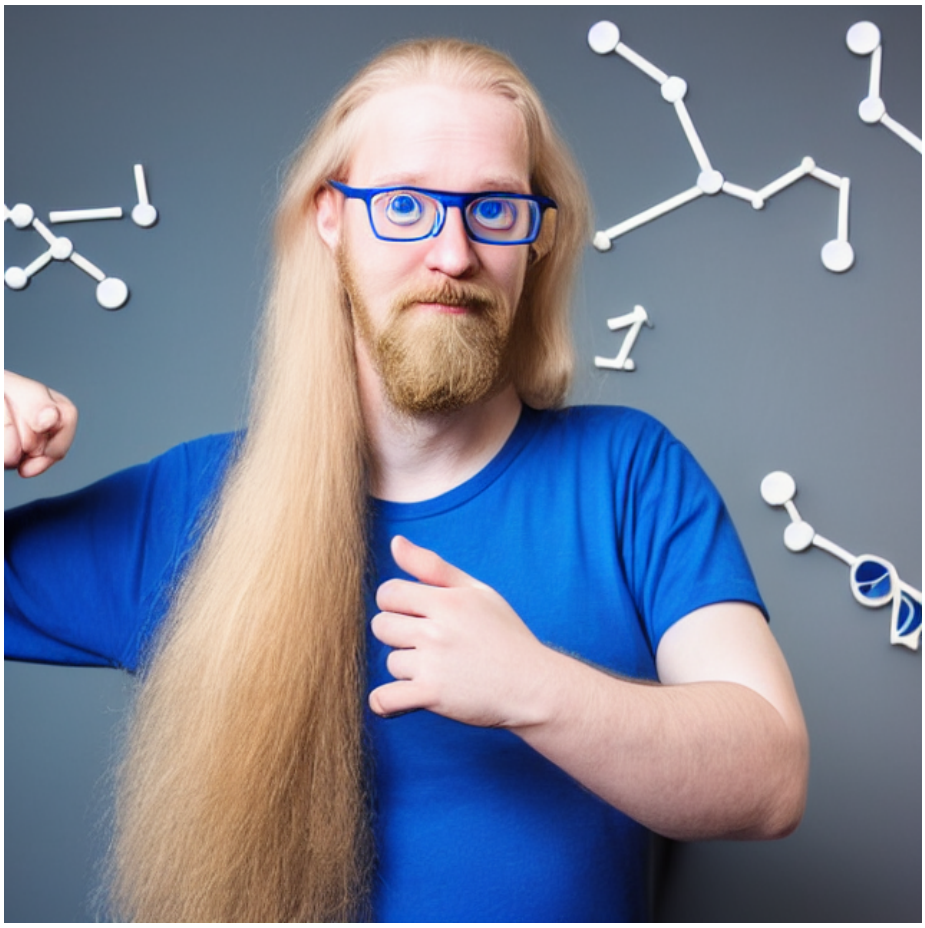
And, finally, I felt that it was only fair to show you the results of my attempts to re-create myself:

Figure 19: “Real Sam”

Figure 20: “Fake Sam”

Figure 21: “Real Veronica”

Figure 22: “Fake Veronica,”



But there's only one original:

Figure 23: “Real Dad,”