Inside the generator

Image generators make a series of numerical choices. A prompt describes part of the target, a random state supplies a starting point, and a trained model guides the output. Changing any of those can change the picture.

This lecture connects those mechanisms to image-making decisions. We will cover diffusion, compressed representations, attention, seeds, guidance, sampling, and editing. The later section extends the same ideas to video, assistants, world models, and robot actions.

The art content is equally specific: camera perspective, depth cues, lightness and color, character continuity, and local image repair. Small calculations will show what a setting changes. Published figures will provide examples, and controlled comparisons will separate a setting effect from a lucky result.

Three questions for today

The first question is why one prompt can produce many images. The answer involves both incomplete instructions and randomness. A sentence rarely specifies every position, surface, and lighting condition, so many images may fit it.

The second question is why stronger guidance can reduce quality. Guidance changes how predictions are combined. It can strengthen requested features while exaggerating unwanted ones or reducing variation. We will calculate a simple example before looking at actual sample grids.

The third question is which control matches a problem. A wrong layout, a damaged local detail, and an inconsistent character are different failures. Seeds, reference images, spatial controls, masks, and trained adapters act at different parts of the system. Knowing that difference makes a test more informative.

Content, layout, and appearance

An image request has several parts. Content names the objects and their relations. Layout specifies where they appear and how much space they occupy. Appearance covers edges, color, light, and material. Intended use determines which details need to remain clear.

For example, a poster may need open space for a title. An animation frame may need a character whose costume matches the previous shot. A small thumbnail may need a simple silhouette. These can use similar subjects while requiring different compositions.

The station image is one example for a few comparisons. Its yellow coat, red prop, and long platform make color, identity, and perspective easy to identify. They are separate variables that can be specified, changed, and checked.

One prompt can produce many pictures

Look at these three pictures for a moment. They came from the same description of an astronaut riding a horse.

The subject stays roughly the same, but many choices change. Look at the horse’s pose, the background, and where the rider sits in the frame.

The words don’t settle all of those choices. So there can be several different pictures that fit the request.

These are older published examples, but the question they raise is still useful: what did the prompt actually specify?

Try describing the differences without saying better or worse yet. One image may place the traveler near the edge. Another may make the platform wider. If we first name the changes, we can later decide which ones matter for the story, instead of choosing only by first impression.

Recognizing an image and making an image

Recognizing a picture and making a picture are different jobs. If I show you a chair, you might say, “That’s a red chair.”

But if I ask you to draw one, you need to choose its shape, the viewing angle, the light, and what sits behind it.

A generator has to fill in those missing choices too.

Recognition can leave many details undecided. A label doesn't need to describe every window in the station. Generation has to put something there, even when we never asked about the windows. This is one reason a short prompt leaves so much room for the model to make choices.

The prompt leaves choices open

“A traveler at a station” leaves a lot open. Are we close enough to see their face? Is it morning or evening? Is the station busy?

The model has learned patterns from training examples, and those patterns influence what it produces. Adding more words can narrow the possibilities.

It still doesn’t describe every part of the picture, and the model may miss some of what we ask for.

Suppose we add the words quiet and lonely. Those words might influence color, empty space, posture, or lighting. They don't specify one exact arrangement. If one arrangement matters, describe it directly. We can then judge whether the image follows that request and whether it creates the intended feeling.

How words and reference images help

Words are useful for saying what we want. A reference image can be useful for showing it.

For example, “make the traveler small” leaves room for interpretation. A sketch with a small figure in the corner gives more direct information about the layout.

We call these extra inputs conditions. They guide generation, but they don’t guarantee that every detail will stay exactly as supplied.

Words and references also have to agree. If the reference shows a close-up while the prompt asks for a distant figure, the system receives competing directions. Before adding more words, check whether your inputs are asking for the same picture. Sometimes simplifying the input makes the test much clearer.

Camera distance changes perspective

Perspective depends on the camera's position relative to the scene. A nearby object occupies more of the image than an otherwise similar object farther away. Moving the camera changes those distance relationships. That is why a close viewpoint can exaggerate a face, a hand, or the front of a building.

In a simple pinhole camera, projected size is proportional to focal length and inversely proportional to distance. If the object and focal length stay fixed, doubling the distance halves its projected size. This is a geometric relationship, not an artistic rule.

Changing the crop at one fixed viewpoint changes framing but preserves perspective. Moving backward and choosing a longer lens can keep the foreground subject similar in size while changing its relation to the background. Generated images may imitate these effects, but a camera word in a prompt does not guarantee calibrated optics.

Three depth cues for a flat picture

A flat image can suggest depth without containing an explicit three-dimensional model. Overlap establishes a front-to-back relationship: if one shape blocks another, we usually read the blocking shape as closer. The cue gives an ordering, not a measured distance.

Relative size supplies another cue when we assume objects have similar real-world sizes. Repeated columns that get smaller can suggest distance. If the objects are actually different sizes, that inference can be wrong. The cue depends on both image geometry and our expectation about the objects.

Linear perspective adds a directional structure. Projections of parallel lines can converge toward a vanishing point. Rails, floor edges, and roof beams can organize a picture around those directions. Different sets of parallel lines may have different vanishing points.

In a generated scene, these cues can disagree. A prop may overlap correctly while its scale changes implausibly. A visually attractive corridor can have inconsistent convergence. Checking overlap, scale, and line directions separately gives a more precise geometric critique than saying the space looks strange.

What to save with an image

When you get a picture you want to keep, save how you made it. The prompt is only part of the record.

You also need the model, any reference images, the seed, and the generation settings.

Think about sending the picture to a classmate and asking them to continue the work.

If all they have is your sentence, they'll have to guess much of the setup.

A model checkpoint just means a saved version of the model's learned values. Keep its name too.

A record is most useful when another person can follow it. Give the image a version number, save the exact prompt, and keep the unedited output as well as your final edit. Otherwise, you may later compare a raw model result with a picture that has already been heavily changed.

Learning and making an image

Before we go further, we need to separate training from making one image. During training, the model’s internal values change as it learns from examples.

During normal image generation, those values usually stay fixed. What changes is the image being generated.

So, if you increase the sampling steps, you haven’t taught the model more about hands or faces. You’ve given the generation process more updates.

That can affect the result, but it’s a different kind of change.

The two stages also run on different timescales. Training can involve a large collection of images and many updates. Making one image uses the trained model for a much shorter computation. When you change a seed in an ordinary generation tool, you aren't asking it to learn the subject again.

What happens when we add noise?

Read this strip from left to right. We start with a recognizable picture and add more and more noise.

Eventually, it becomes difficult to tell what was there. Here, noise means random changes in the image values. We add it deliberately during training.

Why create this problem? Because we have the original picture, and we know the noise we added.

That gives us something we can ask the model to predict, and an answer we can compare its prediction with.

At a small amount of noise, you can still recognize the station. At a large amount, it becomes difficult to say what was there. The learning problem changes with the noise level. Recovering a tiny edge detail is different from making a useful prediction when almost all visual evidence is gone.

Clean image plus noise

This equation describes adding noise to a clean image. Read it as: a weighted amount of the clean image, plus a weighted amount of random noise. The letters a and b control how much of each we use at this noise level. If the image weight is zero and the noise weight is one, we have only noise. If the noise weight is small, more of the image remains visible. This is a useful formula for the diffusion setup we discussed. Other generation methods may describe their states differently.

For one coordinate, suppose the clean value is point eight and the sampled noise value is minus point two. With weights point six and point eight, the noisy value is point six times point eight, plus point eight times minus point two. The result is point three two. These weights have squared values that sum to one, a common normalization for this example. The arithmetic applies to every coordinate, not just one visible pixel.

We can choose how noisy a training image is

We can choose how noisy each training example should be. One example might have a little noise. Another might be almost entirely noise.

We don’t have to create every earlier stage first. We can calculate a noisy example at the level we want.

That lets the model practice across different noise levels. It doesn’t mean that making a finished image will always take only one step.

The noise level is information the network needs. Without it, a small fuzzy patch might be interpreted differently at different stages. Think of the level as telling the model how much damage to expect. It still has to work out a prediction from the image and any other inputs.

How the model learns to predict noise

Let’s follow this figure. Start with the clean picture on the left. Add noise, and give the noisy picture to the network.

In this version of the training task, the network tries to predict the noise we added. Then we compare its answer with the actual noise.

When the prediction is wrong, training adjusts the model’s internal values. This happens over many examples. The model also gets information about the noise level.

I’ve left that input out of the drawing so the main sequence is easier to see.

Across training examples, the network sees many ways that useful image structure survives under noise. It changes its internal numbers when its prediction is wrong. No one writes a separate rule for every coat or platform. The behavior comes from learning patterns across the examples it was given.

How we measure a prediction error

During noise-prediction training, we know the noise that was added. We subtract the model’s prediction from that known answer, square the differences, and average them. Squaring makes large errors count more strongly. Training tries to bring this error down across examples. Notice what this score measures. It measures the noise-prediction error. It doesn’t directly score whether the picture tells a good story or whether its composition suits this image. Those need separate judgments.

For a two-coordinate example, let the known noise be one and minus one. Suppose the prediction is point eight and minus point five. The errors are point two and minus point five. Squaring gives point zero four and point two five, and their mean is point one four five. Training uses many coordinates and examples, but this small calculation shows exactly what the error score measures.

Where does the finished image come from?

There’s an important difference between this training example and generating a new picture. During training, we began with a known image.

During generation, we can begin with random noise. There isn’t a particular photograph hidden inside it, waiting to be uncovered.

The model uses patterns it learned to help produce a possible image.

That doesn’t rule out copying or memorization; it just explains what this generation process is doing.

This is why generation is more than uncovering a picture that was waiting inside the noise. At the start, many outcomes are possible. The learned model and the inputs guide a sequence of choices. By the end, those choices have become one particular arrangement of objects, surfaces, and light.

Discussion: training or sampling?

Separate these two actions. In the first, a model is trained using a new collection of images. In the second, the same trained model generates another image with a different seed. Which values change in each case?

Training changes learned weights. Generation changes the current sampled state while normally keeping those weights fixed. A new seed changes the random starting process. It does not add the new output to the training set or teach the model a new subject by itself.

This distinction matters when choosing an intervention. A different sample may solve a particular composition by chance. Subject adaptation aims to change learned behavior across future samples. They have different costs and different purposes. The later LoRA and reference-conditioning examples will make that difference more specific.

What changes during generation?

The main distinction is this: training changes the model. Sampling uses the model to change the current image.

More steps may change how the result develops, but they don’t directly tell the model to move the traveler farther away.

For that, a wider-view instruction or a layout reference may be a better thing to test.

A sharper version of the wrong composition is still the wrong composition.

The word update can make these stages sound alike. In training, it means changing what the network has learned. In sampling, it usually means changing the current state being generated. Keep asking what is being updated. That small question prevents a lot of confusion when you read a technical diagram.

From noise to an image

Now let’s run the process toward an image. In this diagram, the noisy state is on the left and the cleaner image is on the right.

We start with noise, ask the model for a prediction, and use that prediction to make an update. Then we repeat.

You don’t need to follow every symbol here. Follow the changing state.

Each prediction depends on what is currently there and where we are in the process.

There is no finished target picture supplied at the end for the model to copy.

Don't expect every intermediate image to be a useful draft. Some states are hard to interpret, especially early on. For a design review, finished samples are often easier to compare. Intermediate pictures are useful when we specifically want to understand how the computation changes over the course of a run.

The model predicts. The sampler takes a step.

The model and the sampler have different jobs. The model makes a prediction from the current noisy state.

The sampler uses that prediction to calculate the next state.

That calculation is why changing the sampler can change the result even when we keep the model.

We’re using the same learned network, but changing how we follow its predictions.

Imagine the network says which direction to move, while the sampler decides how to take that move. If we change the sampler, the network can remain the same, but the route through intermediate states can differ. So two tools can use similar models and still behave differently during generation.

Different steps use different noise levels

The amount of noise changes during generation. Near the noisy end, very little image information is visible.

Near the end of the process, the picture is easier to recognize. A schedule chooses which noise levels we visit.

So step ten in one setup may not mean the same thing as step ten in another.

Don’t assume every model follows a fixed “composition first, details later” sequence.

Equal numbers of steps don't necessarily mean equal spacing along the noise levels. One schedule may spend more updates near a particular part of the process. That changes where the model's predictions are evaluated. When a tool offers a schedule option, it is changing more than the label on the step counter.

What can the model predict?

We’ve explained this using a model that predicts noise. That’s one common training choice.

Other models can predict a clean image, or a direction in which the current state should change. You don’t need to memorize all the versions.

The useful point is that the sampler has to use the prediction in the way the model was trained for. These outputs aren’t interchangeable labels.

These prediction targets can sometimes be converted into one another when we know the noise level and the formula connecting them. But the network was trained for a particular target. Copying a formula from another model without checking that target can give an update that doesn't mean what you think it means.

Working with a smaller representation

Working directly on every pixel can take a lot of computation. Some models do most of the work in a smaller representation of the image.

That representation is called a latent. It’s a collection of learned numerical values.

In this older Stable Diffusion example, the image and its working representation have the sizes shown here. There are far fewer values in the smaller one.

That reduces the amount of information the image model has to handle, though it doesn’t give us an exact speedup from the numbers alone.

The smaller representation saves work, but it changes where that work happens. We still need an encoder for images we supply and a decoder for visible output. A fast denoising loop doesn't remove those other parts. For an interactive artwork, measure the whole wait that a visitor actually experiences.

How an image gets compressed

The encoder turns an image into that smaller representation. The decoder turns it back into pixels. Look at the middle of the figure.

The little grid is just a way to show that the model works with numbers.

It isn’t actual data from this picture, and there isn’t one square meaning “coat” and another meaning “suitcase.”

In a latent diffusion model, the repeated generation steps happen in this smaller form. We use the decoder to get the image we can see.

A useful way to picture the decoder is as a learned way of turning compact information into visible structure. It doesn't simply enlarge each colored square in this drawing. The actual representation spreads information across numbers, and the decoder combines them using patterns learned during its own training.

Rebuilding an image and making a new one

Rebuilding an existing image and making a new image test different things.

If I encode this picture and decode it again, I can check how much survives. That tests the compression part.

It doesn’t show that any random set of numbers will turn into a good picture.

The generation model still has to produce a useful representation for the decoder to work with.

Try the two checks separately. First, pass an existing image through compression and reconstruction. Then examine newly generated samples. If the first test loses thin lettering, that gives one possible cause of poor lettering in the second test. It doesn't explain every failure, but it narrows the question.

What does a VAE add?

A VAE is a variational autoencoder. The full name matters less than the extra idea it adds to compression. Its encoder describes a range of possible values for the smaller representation. Training asks it to reconstruct images while also keeping those values organized in a way that supports sampling. Those two aims can pull in different directions. Keeping every detail and making a useful space of representations aren’t exactly the same task. Image models can use related autoencoders with additional training losses, so this is an introduction to the idea rather than a description of every system.

The reconstruction term asks whether the decoded image resembles the input. The regularization term compares the encoder's distribution with a chosen prior distribution. That second term discourages every image from being encoded into an arbitrary isolated region. The balance affects the usefulness of the latent space. In image-generation systems, additional perceptual or adversarial losses may change the reconstruction behavior, so the simple VAE objective is a starting model of the idea.

Which details survive compression?

Think about a thin railing, a tiny logo, or the handle of our suitcase.

If that detail disappears when we simply encode and decode the picture, changing the prompt won’t tell us much about the cause.

We already know some information is being lost in that part of the system.

It helps to check where a problem appears, rather than treating every failure as a wording problem.

This matters for work that people view at different sizes. A tiny suitcase handle might disappear in a projection even if it survived generation. Check the actual display size as well as the original file. Image quality is partly about whether the important details remain visible where the work will be shown.

Edges and texture: paper, paint, and ink

Edges help organize an illustration. A hard edge creates a clear boundary. A soft edge allows neighboring shapes or tones to merge. A lost edge occurs where the boundary becomes difficult to separate from its surroundings. Artists can use all three within one image.

The cut-paper example emphasizes separate shapes. The paint example uses softer transitions. The ink example emphasizes line weight and marks. These differences affect which forms remain readable at a small size and which areas attract attention through contrast.

A style prompt can influence several of these properties at once. If the goal is a clear silhouette, simply adding more surface detail may work against it. Specify the edge behavior or value separation that the image needs. The three illustrations are teaching examples, not controlled measurements of a style setting.

The parts of an image generator

Here is how the parts fit together in one image generator. The random starting values come from the seeded random-number generator.

The text encoder turns the prompt into information the image network can use. The network makes predictions, and the sampler applies updates.

At the end, the decoder produces pixels. Follow the loop in the middle. That’s the repeated work. Later models can replace some of these parts.

This diagram is useful because it shows where the controls enter, not because every current product has exactly this structure.

When a result fails, this diagram gives us places to investigate. A prompt misunderstanding may involve text conditioning. Lost tiny detail may involve compression or decoding. An unstable comparison may involve the random setup. We won't diagnose everything from one picture, but we can choose a more focused next test.

How a model uses words

The model can’t use a sentence in the same form that we read it. First, the sentence is divided into small pieces called tokens.

A token can be a word or part of a word. A text encoder turns those pieces into numerical features. The image model uses those features.

But having information about “red” and “suitcase” doesn’t guarantee that red will end up on the right object.

Adding words also changes the input representation. More words don't always make the request clearer. If several phrases repeat the same idea while another phrase conflicts with it, the result can be hard to interpret. Start with the subject, placement, and appearance that matter, then add a detail for a reason.

How CLIP links pictures and words

CLIP is one way to connect words and pictures. It learns from pictures paired with captions.

During training, it learns to give a matching picture and caption similar numerical descriptions, compared with pairs that don’t match.

Some image generators use CLIP’s text encoder to help interpret a prompt. Other systems use different encoders. CLIP itself isn’t drawing the image.

It helps connect what the words describe with visual information. The generator still has to make the picture.

Matching pictures and captions teaches useful associations, but the captions are not complete descriptions of everything in a picture. They may mention the person and leave out the railing. That helps explain why a system can recognize the overall topic while missing a precise relation that was never easy to describe.

A similar-looking image can still be wrong

A picture can be broadly similar to the prompt and still be wrong in a way that matters.

We might get a station, a traveler, and a suitcase, but the suitcase could be on the wrong side.

Or we ask for two objects and get three. So don’t stop at “it looks about right.” Check the particular details the work depends on.

For a poster, a small relation can decide whether the image is usable. A hand holding a ticket is different from a ticket floating beside the hand. When you write your checks, include the relations that carry the meaning, not only a list of nouns that should appear somewhere.

How attention shares information

Attention is a way for the model to use information from different places. With self-attention, parts of the same input share information.

One area of an image can use information from another area. That can matter for things like an object and its reflection.

With cross-attention, information comes from another input. An image area can use information from the prompt.

The names sound complicated, but the distinction is simple: information from within the same input, or information from another input.

Attention doesn't make the model conscious of an object. Here it names a calculation that selects and mixes information. The useful design connection is that a local feature can depend on distant features or words. Changing one instruction can therefore influence more than the small area you had in mind.

An image area can use information from words

This drawing gives us a rough picture of cross-attention.

The area around the coat can use information linked to “yellow,” along with other words in the prompt. Different information receives different weights in the calculation.

The lines here are drawn to explain the idea.

They aren’t measurements from a model, and we can’t use them to prove which word caused a particular pixel.

What matters for now is that the text can influence image features during generation. It isn’t just read once and then put aside.

The word coat does not need to control only the pixels inside the coat. It can influence shape, shadows, and nearby areas through later layers. A cross-attention map is therefore a clue about a calculation, not a clean selection mask. If we need to edit just the coat, an explicit region mask gives a different kind of control. That distinction matters when a small wording change unexpectedly alters the face or the background.

Attention as a weighted mix

Here is attention as a weighted mix. Suppose the weights are point seven, point two, and point one. We multiply each corresponding set of features by its weight, then add the results. The weights here sum to one. The model is mixing numerical information. It isn’t necessarily choosing one word and ignoring all the others. These numbers are invented to make the arithmetic easy to follow. They aren’t measurements from the coat image. In a real network, many such calculations happen across different parts and layers.

Using scalar values makes the mixing easy to calculate. Let the three values be ten, twenty, and thirty. With weights point seven, point two, and point one, the result is seven plus four plus three, which is fourteen. A neural network usually mixes feature vectors instead of these three scalars. The weights come from comparisons involving queries and keys; the weighted information comes from the values.

Why position matters

Order matters. “Dog bites person” and “person bites dog” use the same words but describe different events. Position matters in images too.

A suitcase on the viewer’s left isn’t in the same place as one on the right. The model needs information about word order and image position.

Having that information helps it represent relationships, though it can still get a relationship wrong.

For a camera, left and right are usually screen coordinates. For a character, left and right can refer to their own body. A character facing us has their right hand on our left. This simple reversal is a common source of ambiguous instructions. Write the viewpoint into the request when it matters, such as the suitcase on the viewer's left, held in the traveler's right hand. Then check those two relations separately.

Discussion 2: Does it match the instructions?

Here’s our next check. We want a yellow coat, a red suitcase on the left, and the station roof on the right.

With a partner, make a short list of what you would inspect in the result. Be specific enough that another person could use your list.

Now compare your lists. Could an image look beautiful and still fail one of those checks?

Also, when we said “on the left,” whose left did we mean?

A useful answer could contain four checks: one traveler, a yellow coat, one red suitcase, and the roof on the requested side. A fifth check could describe the relation between the hand and handle. These are observable requirements. Quietness is different: we might discuss empty space, low contrast, or posture as evidence. Keeping those categories separate lets a group disagree about mood without losing track of a clear object error.

Hue, lightness, and figure–background separation

Hue and lightness describe different properties of color. Yellow and red name different hues. Lightness describes how light or dark a color appears. Two visibly different hues can still be similar in lightness, making their boundary less clear in some viewing conditions.

A grayscale version can help inspect the value structure of a composition. If a coat and the wall behind it become nearly the same gray, the silhouette may need another form of separation. Possible changes include a darker background, a lighter figure, or a controlled edge of light.

This is a diagnostic, not a rule that every artwork must have strong contrast. Low contrast can be intentional. The useful distinction is between an intended soft boundary and a boundary that disappears accidentally. More saturated color alone does not necessarily solve a value problem.

A transformer can remove noise

A transformer can be part of a diffusion model. Look mainly at the left side of this figure. The noisy representation is split into patches.

The patches become tokens, pass through transformer layers, and are turned into a prediction. “Transformer” names the kind of network. “Diffusion” describes the generation approach.

They can work together. This published diagram used class labels, such as an object category. It wasn’t originally a text-prompt model.

We’re using it to understand the network structure, rather than every detail of its original task.

The patch sequence carries both image information and position information. Without position, the same collection of patches could describe several different layouts. Transformer layers then exchange information across that sequence. This is why a transformer denoiser can connect distant parts of the picture. But full attention becomes expensive as the sequence grows. The cost is especially important for video, where adding frames also adds a time dimension to the tokens being processed.

Smaller patches mean more work

Smaller patches give the model more tokens to process. In full self-attention, each token is compared with every other token.

That means the number of comparisons grows quickly. Twice as many tokens gives four times as many pairwise scores.

That doesn’t mean the whole program always takes exactly four times longer. It explains why this part of the calculation can become expensive.

Take a square latent grid with sixty-four positions along each side. With four-by-four patches, it becomes sixteen by sixteen, or two hundred and fifty-six tokens. With two-by-two patches, it becomes thirty-two by thirty-two, or one thousand and twenty-four tokens. That is four times as many tokens and sixteen times as many pairwise scores in full attention. These numbers describe one attention calculation, not the total runtime of the model.

The network and the generation method

There are also different ways to generate the output. An autoregressive model predicts the next token using the tokens that came before it.

You've seen that idea with text: "The cup is on the…" and then a possible next word.

A diffusion model instead updates a noisy state over several steps. Both can use transformers.

So when you read a model description, separate the network it uses from the order in which it generates the result. Those names answer different questions.

Think of these as answers to two different engineering questions. The network describes how information is processed at one call. The generation method describes how those calls are arranged to produce an output. A transformer can predict the next text token, the next image token, or an update to a noisy image. Seeing the word transformer on a model page is not enough to know which process the product uses.

Making an image one token at a time

An image can also be represented as a sequence of tokens.

One kind of image generator predicts those tokens one after another, using the earlier ones as context.

Once it has the sequence, another part turns it into pixels. That’s the basic idea here.

It doesn’t mean all image generators draw from the top-left corner to the bottom-right, pixel by pixel.

A sequence model learns a conditional distribution for the next token. At each position, it assigns different probabilities to possible choices. Sampling selects among those choices according to a rule. Taking the highest-probability token every time is another rule and can reduce variation. In image systems, the tokens may describe learned visual units rather than ordinary words. The order of prediction and the meaning of the tokens both matter.

Flow matching: learning how to move

Flow matching gives us another way to describe learning a generation process. For this simple training example, imagine a path joining noise to a known image. We can pick a point on the path and ask the model to predict the direction of movement there. At generation time, the final image isn’t supplied. The model has learned directions from many training examples, and we follow its predictions. In this equation, time starts at noise and ends at the image. That is the opposite direction from the earlier equation where we added noise to an image. Keep that change of convention in mind.

For a straight training path, write the state as one minus time multiplied by the starting noise, plus time multiplied by the target data. Differentiating with respect to time gives target data minus starting noise. That supplies a known direction for a training example. The learned field combines information across many examples; it is not handed the desired final image during generation. The choice of path and target is part of the training design.

Following the model’s direction

Once the model predicts a direction, we need a method for following it. A simple update takes the current state and adds the predicted direction multiplied by a small step size. Then we ask for another direction from the new state and repeat. That is the idea behind the example on this slide. Real systems can use more involved solvers. The important point is the same one we saw earlier: the model makes a prediction, and a numerical method uses it to calculate the next state.

Suppose the current coordinate is two, the predicted velocity is three, and the step size is point one. Euler's update gives two plus point one times three, or two point three. Then the model evaluates the direction again at the new point. A large step can miss curvature in the field. Smaller steps can reduce that numerical error, but they require more evaluations and cannot remove an error in the learned direction itself.

Where does randomness enter?

Randomness can enter in more than one place. There may be random noise at the start. Some samplers also add random noise during the run.

Others follow a fixed calculation once the starting state is fixed.

So a fixed sampling procedure can still produce different images if it begins from different starting values.

We need to know the full setup to repeat a result.

We can distinguish random initialization from random movement. An ordinary differential equation sampler can start at random noise and then follow a deterministic path. A stochastic sampler also injects randomness along the path. Both can make varied images. Deterministic here describes the path after the initial state is fixed; it does not mean that every run must start at the same state or produce the same picture.

Four terms that describe different parts

This table puts the terms in order. Transformer or U-Net tells us about the network. Pixels or latents tells us what representation it works on.

Diffusion or flow matching tells us about the generation training approach. The sampler or solver tells us how updates are calculated.

Take a moment and explain one row to the person beside you. You don’t have to remember every name today.

I want you to recognize that these are different parts of a system, rather than competing names for the same thing.

A useful model description might say: transformer network, latent representation, flow-matching training, and a particular numerical solver. Those four phrases can all be true at the same time. Another model can keep three of those choices and change the fourth. This vocabulary helps us compare systems without assuming every new name describes a completely new invention. It also explains why a setting borrowed from one system may not transfer directly to another.

Different seeds can give different pictures

A seed sets the starting point for a repeatable sequence of random numbers.

Changing it can give us a different starting state, and that can lead to a different picture.

These illustrations show the kind of composition changes we might care about. They aren’t measured outputs from particular seed values.

A seed number has no fixed artistic meaning. Seed forty-two doesn’t mean “wide shot,” and forty-three doesn’t mean “slightly wider.” Use seeds to explore alternatives.

Use instructions or references when you need a particular layout.

A seed is an address into a repeatable random process, not an instruction about appearance. Adjacent seed numbers do not have to produce adjacent compositions. If we want to study guidance, the useful arrangement is several guidance values crossed with the same seed list. Each seed then supplies a matched starting condition. If we only change the seed until one image looks good, we have selected a result rather than measured the effect of guidance.

How to repeat a result

To repeat a result, keep more than the seed. Save the model, prompt, references, and other settings.

If you’re doing a comparison in code, reset the random generator before each matched run. Otherwise, it may simply continue to the next random numbers.

Exact matching can also depend on the software and hardware. The seed is part of the record, not the whole record.

Software can change the sequence even when a displayed seed stays the same. A new sampler, another model revision, or a different random-number implementation can alter the computation. For a class experiment, a saved workflow file is often more useful than a screenshot of the seed box. Record the model version and the full input settings, then keep the actual outputs together with that record.

Character continuity: shape, costume, and props

For a story, we often need the same character in several shots.

Decide which features must stay: the coat, the hair, the suitcase, perhaps which hand is holding it.

The camera angle and pose can change while those features remain recognizable. A repeated seed doesn’t provide a full character description.

References can help, but we still need to compare the actual images and catch changes that break the scene.

Character continuity has several levels. The silhouette can stay recognizable while the face changes. The face can remain similar while the coat gains an extra pocket. A prop can keep its color but switch hands between shots. A character sheet makes those separate requirements visible. It is especially useful before a sequence is generated, because it defines what must remain stable instead of leaving every shot to invent those details again.

A character sheet defines what must stay

A character sheet defines the stable features of a subject across views. This illustration shows one costume and prop from three directions. Its purpose is to make the requirements visible before generating a sequence, rather than inventing those requirements after the shots are complete.

The silhouette includes coat length, shoulder shape, and overall proportions. Surface details include buttons, pockets, hair, and the prop's shape. Ownership includes which hand carries the prop. These features have different visibility in a front view, side view, and back view.

A reference method can help preserve these features, but the reference should also be internally consistent. If two source images disagree about the costume, the generator receives conflicting evidence. A clean reference set reduces that ambiguity. It does not guarantee that every new view will be correct.

For evaluation, distinguish legitimate view changes from identity drift. The visible side of a suitcase should change when the camera moves. Its handle should not switch construction for no reason. This distinction allows variation while keeping the subject recognizable.

Guidance: how strongly to follow the prompt

Now we can look at guidance. In classic classifier-free guidance, we compare a prediction made with the condition to one made without it.

The condition might be a text prompt or a category label. Guidance pushes the result further in the direction of that difference.

This can make the requested subject more obvious. But turning up the strength doesn’t make the instruction more precise.

If the model’s direction is imperfect, a stronger push can make that problem more visible too.

During the classic training setup, the condition is sometimes dropped. That allows the same network to learn predictions with and without that information. At sampling time, the difference between those predictions supplies a direction for guidance. This connects a training choice to a user-facing control. The strength slider is not a universal measure of how well a model understands language; it controls a particular combination of predictions.

How guidance combines two predictions

You can read this formula in words. Start with the prediction without the prompt. Then add a scaled difference between the prompted and unprompted predictions.

At a scale of one, this formula gives us the prompted prediction. Above one, we push beyond it. We aren’t simply averaging two answers.

Different papers and tools sometimes use different numbering.

In the paper we’ll use for the later exercise, their guidance number is shifted by one compared with this formula.

That’s why we should check the definition before comparing values.

The direction in this formula lives in a high-dimensional numerical space. It is not a single slider for yellow, sharpness, or realism. Several visual properties can change together when the scale changes. The one-number example on the next slide is useful precisely because it shows extrapolation clearly: we start from one prediction and continue past the other. In a full image, that happens across many coordinates at once.

A small guidance calculation

Let’s make the guidance formula concrete with one number. Suppose the unprompted prediction is two and the prompted prediction is three. Their difference is one. At guidance scale one, we get two plus one, which is three. At scale four, we get two plus four, which is six. We have moved beyond the prompted prediction rather than taking an average between two and three. Those are made-up values for one coordinate, not actual image measurements. They show why a larger guidance scale can exaggerate a direction.

The scale-four result is outside the interval between two and three. That is why guidance can amplify a feature rather than merely blend two plausible answers. The numerical space has many coordinates, so several properties can be amplified together. A lower scale can be useful when that amplification damages color or detail. The actual setting still depends on the model's training and the exact guidance convention used by the software.

What can go wrong with strong guidance?

Look at what happens to appearance when guidance becomes stronger. The subject may be easier to recognize.

But colors can become too strong, some details can look strange, and different samples may become more alike. Whether that is useful depends on the work.

A bold poster and a quiet scene may need different choices. These are examples from a published study using image categories.

They show a tradeoff worth checking. They don’t give us one best number for every current image model.

In an illustration, exaggerated contrast can make the silhouette read clearly at a small size. The same contrast may destroy a soft transition needed for fog or dusk. We should therefore distinguish an artifact from an intentional graphic effect. The problem is not strong color by itself. The problem is losing control over which shapes are emphasized and whether the effect remains consistent with the intended image.

How a negative prompt can help

A negative prompt changes the comparison in some image generators.

Instead of comparing with an empty prompt, the system can compare with the words in the negative prompt.

The calculation then pushes away from that prediction and toward the positive one. But it isn’t an exact removal tool.

Writing “people” in a negative prompt doesn’t guarantee an empty room. If something unwanted remains, inspect it and try a clear change.

Adding a very long list of negative words can make it harder to tell which part helped.

A negative prompt changes a conditioning input; a mask defines a spatial region. They solve different problems. If the unwanted element is one sign in the corner, a local edit may give more direct control than a long list of negative terms. If the unwanted tendency appears across the entire picture, changing the prompt or the model settings may be more relevant. Choose the control that matches the location and scale of the problem.

Three guidance settings

Here are three published examples at different guidance settings. The request describes an astronaut in a jungle, with cold, muted colors.

Look at the color, the size of the subject, and the feeling of the surroundings.

There’s something missing from this comparison: the displayed examples don’t document matched seeds.

So we can describe the differences, but we can’t confidently say that guidance caused every one of them. Keep that distinction in mind.

A useful-looking comparison still needs a clear record of what changed.

This is also a useful lesson about published figures. A page can show striking differences without providing every setting needed to reproduce them. We should not fill those gaps with assumptions. We can say that one example has stronger color or a larger subject. To attribute those differences to a single variable, we would need the missing controls. Description is valid evidence of appearance; causal explanation requires a stronger comparison.

Discussion 3: Clear subject, wrong mood

The coat is yellow, but the picture feels harsh and crowded.

It shows the right object, yet the mood is wrong. With a partner, describe how you would test different guidance settings. What would you keep fixed?

What would you look for in the results? Now talk about judgment. How would you explain that one version feels quieter or less crowded?

Name something visible, such as contrast, figure size, or the space around the traveler.

Consider two possible changes. Lowering guidance tests whether the harsh colors depend on that prediction combination. Making the traveler smaller tests composition instead. If we do both at once, the result may improve, but the explanation becomes unclear. A useful pair exercise is to give each person one of those changes, keep the starting conditions matched, and compare which visible problem each change actually addresses.

What should we compare?

For that test, keep the model, prompt, image size, sampler, and step count fixed. Change guidance, and use the same set of seeds at each value.

Several seeds are better than one, because a setting may work well for one starting point and poorly for another.

Then compare separate things: whether the image shows what we asked for, whether we like its appearance, and whether the set gives us useful variety.

You may find that no single setting wins on every question.

A small two-factor experiment might use three guidance values and four seeds, giving twelve images. Every guidance value appears with every seed. That layout separates a setting effect from a lucky starting point more clearly than three unrelated favorite images. We can then mark prompt errors, visible artifacts, and composition changes in separate columns. Twelve is only a classroom example, not a statistically sufficient sample for every research claim.

15 steps and 50 steps

Now compare fifteen steps with fifty steps. These are two completed runs from an older Stable Diffusion example. The prompt and seed were kept the same.

They aren’t two snapshots taken during one run. Look at the horse and the astronaut’s clothing. What changed? Which changes would you actually call improvements?

More steps give a different generation path. They can help, but they don’t guarantee that every object becomes more correct.

And this older example doesn’t tell us that a newer model should use fifty steps.

More updates can reduce numerical approximation error for a suitable solver and model. But the learned prediction can still be wrong. Taking a more accurate route through an imperfect learned field does not guarantee the object count or anatomy is correct. This separates two sources of error: how well we follow the model, and how well the model represents the desired images. Increasing steps mainly addresses the first computation, not every limitation of the second.

Steps and sampler: two different settings

Step count tells us how many updates to take.

The sampler tells us how to calculate those updates, and the schedule tells us which noise levels to visit.

Those are related choices, but they aren’t the same choice. Think about following a route.

The number of stops doesn’t tell you where the stops are or how you travel between them. If you’re comparing speed, measure the actual time.

A displayed step can involve different amounts of work in different systems, so step count alone isn’t enough.

Some numerical methods ask for more than one model prediction to calculate a single update. Others reuse information from earlier predictions. This is why a comparison based only on a visible step count can be misleading. For an experiment, report both the settings and elapsed time. For a software implementation, the number of network evaluations is another useful measure because the network often accounts for much of the work.

A progress picture or a finished result?

A progress image comes from partway through one run. A step-count comparison uses separate runs that each finish.

That difference matters because changing the total step count may also change the schedule from the beginning.

A finished fifteen-step image isn’t necessarily what a fifty-step run looks like after its fifteenth step.

So, when you show a comparison in your work, label it clearly. Are we watching a single image develop, or comparing finished results from different settings?

Those pictures answer different questions.

Imagine one run visits twenty noise levels and another visits fifty. Their fifteenth updates may occur at different noise levels, so comparing those intermediate states does not compare equal points in the process. A progress sequence should identify its own run. A final-result grid should identify the full settings of each completed run. This makes the figure answer a clear question rather than mixing two different kinds of evidence.

Generation cost: count model evaluations

Step count is not always the same as computational cost. In classic classifier-free guidance, one update uses a conditional prediction and an unconditional prediction. With twenty sampling steps and one pair per step, the simple count is forty network evaluations.

An implementation may batch the pair together. That can reduce elapsed time compared with two separate calls, but it does not turn the calculation into one unconditioned prediction. A different solver may request more evaluations, while a distilled model may implement guidance differently.

For an interactive display, the relevant time is the complete wait from input to visible result. It includes text encoding, sampling, decoding, and any additional processing. Image size, model size, and hardware also affect that wait. Report measured seconds when comparing speed, alongside the sampler and step settings.

Some models need only a few steps

Some models are trained specifically to work with very few steps. SD-Turbo is an example.

Its documented text-to-image setup uses one to four steps and turns off the usual classifier-free guidance.

That is a property of this trained model and its intended setup.

We can’t take those settings and assume they work the same way in an older model.

The training process called distillation helps a model learn a shorter generation process.

For our purposes, remember this: a model trained for one step is different from simply stopping another model after one step.

Distillation changes what the model learns to do in a limited number of calls. It may train a student using information from a more expensive teacher or use related objectives to shorten generation. The resulting model can have a different recommended sampler and guidance behavior. We should use its documented operating range first. A one-step model and a fifty-step model are different trained systems, not simply the same system with different patience.

Changing the model

Changing the model can change much more than the appearance. The training images, network, text encoder, and training method may all differ.

Even a setting with the same name may behave differently. For your own work, decide what you're comparing.

Are you asking which complete tool helps you make the scene? Or are you asking which technical change caused an improvement?

The first is a useful design question. The second needs a more careful experiment.

One favorite picture from each model won't tell us very much about either.

There is also a difference between comparing models and comparing complete workflows. A workflow may include a better reference interface, automatic resizing, a face repair stage, or an upscaler. Those extra stages can matter to a designer even if the base model is unchanged. If the question is which tool helps finish the poster, include them. If the question is which network change improved quality, separate them from the model comparison.

LoRA: change fewer model parameters

LoRA represents a weight update as the product of two smaller matrices. The base weight remains available, and the learned product supplies an adjustment. The rank sets the intermediate size of that factorization. A smaller rank reduces the number of values being trained, but also limits the form of the update.

For one weight matrix of one thousand by one thousand, a full update contains one million values. With rank eight, the two factors contain eight thousand values each. Their total is sixteen thousand, which is sixty-two and a half times fewer for this particular matrix.

That calculation does not describe every parameter in a real model. It explains the saving for one adapted weight. A style or subject LoRA must still match the base model it was trained for. Rank, training data, and where the update is applied can all affect what it learns.

Reference input or learned subject adaptation?

Reference conditioning and subject adaptation are different approaches. IP-Adapter supplies image features through a learned image-conditioning pathway, with text and image attention separated. Once the adapter is trained, supplying a new reference image does not require training a new adapter for that subject.

DreamBooth instead fine-tunes a text-to-image model using examples of a particular subject. It associates that subject with a special identifier so the subject can be requested in other contexts. The training step is part of the method, rather than just a reference supplied during one generation.

LoRA names a compact way to parameterize a weight update. It is not itself a particular character or style. These ideas can be combined, so the categories are not always mutually exclusive. The important distinction is which information enters during generation and which information has changed the learned weights.

ControlNet: edges, depth, and pose

ControlNet adds spatial conditioning to a pretrained text-to-image diffusion model. The original paper studies controls such as edges, depth, segmentation, and human pose. These maps specify structure in a form that is more explicit than a general appearance description.

An edge map can preserve a building outline without fixing its material. A depth map can communicate near and far surfaces without specifying exact color. A pose map can locate joints while leaving costume and facial detail open. The choice of map depends on which property needs control.

The added network is trained to use that spatial information alongside the text condition. A control image is not simply pasted into the output. It affects the learned generation process. Structure can become easier to specify, while identity, texture, and fine geometry still need their own checks.

Starting with a drawing

We don’t always have to begin with random noise. We can begin with a drawing.

In one common approach, the system represents the drawing, adds some noise, and then generates from that starting point.

The original image still gives it some structure to work with. One stays close to the starting layout. Another makes larger changes.

Before choosing a result, decide what you wanted to preserve. Was it the pose? The roof line? The empty space?

A more detailed image may still lose the part of the drawing you cared about most.

A sketch is especially useful for shape and placement because it can show relations that would take many words to describe. The outline of a roof, the size of the figure, and the gap between the suitcase and bench can be visible at once. However, a rough sketch may not specify lighting or material. The generator still makes choices in those areas. Preserving structure and inventing appearance are therefore different parts of this task.

Editing strength: preservation and change

In this kind of image editing, less added noise often leaves more of the starting image in place. More noise usually allows larger changes.

But the prompt and model still matter, and the meaning of a “strength” setting depends on the tool.

For our sketch, I would first decide what must stay.

If the traveler’s position is important, I’d compare that position across results before judging the clothing texture.

That gives me a clear reason to keep or reject an edit, instead of just choosing the most detailed one.

There is a tension between fidelity to the starting drawing and freedom to change it. If the starting pose is wrong, preserving it too strongly can preserve the mistake. If the pose is right, adding too much noise may remove the feature we wanted to keep. A useful comparison uses the same drawing at several strengths and names the exact structures to preserve before choosing a result.

Local editing starts with a mask

A mask specifies where an image edit is allowed to appear. In simple compositing, a value of one selects the edited image and a value of zero selects the original. Values between zero and one blend the two. The formula on the slide describes this blending operation.

For a pixel with a mask value of one quarter, the composite uses one quarter of the edit and three quarters of the original. This can create a soft transition at an edge. But a wide soft boundary can also create a halo or mix incompatible lighting.

An inpainting model uses a region specification as part of generation. That is different from directly compositing a finished edit with the original pixels. Some tools may alter unmasked areas internally. If exact preservation matters, compare the output outside the region or explicitly retain the original there.

The art problem remains at the boundary. The repaired region needs compatible perspective, edge treatment, color, and light. A technically correct mask can still produce an obvious pasted-on result if those properties do not agree.

A layered image-editing workflow

A layered workflow assigns different tasks to different operations. Composition can begin with a sketch, depth map, or other spatial reference. Image generation supplies appearance and detail. A local mask can repair one region. Color adjustment and typography can remain in editable layers.

This is particularly useful when exact lettering or alignment matters. A poster title can be typeset independently of its background. An edited prop can be composited over a generated base. The final image may therefore combine generated and directly controlled elements.

The technical connection is preservation. If every revision regenerates the whole image, all regions can change. A layered edit limits the scope of the revision. It still requires checking edge seams, lighting, and color consistency. Keeping the base, masks, and final composite makes the work easier to revise later.

A video needs to stay consistent

A video adds another problem: things need to remain consistent over time. In the top row, the coat and suitcase keep their appearance across shots.

In the bottom row, they change. Each frame could look acceptable on its own, but together they break the scene.

Video models can use information across frames.

Even so, you still need to watch for disappearing objects, changing faces, or walls that bend during a camera move.

These rows are illustrations of those problems. They aren’t test results from a particular video model.

The question is what we would check when watching a real output.

Temporal consistency has several meanings. Object identity concerns whether the same person or prop stays recognizable. Geometric consistency concerns whether surfaces keep their shape and position. Motion consistency concerns how positions change between frames. A model can succeed at one and fail at another. For example, a face can remain recognizable while the wall behind it bends. Separating those errors makes a video comparison more informative than a single realism score.

2026: Text inside pictures

Let’s look at some examples from 2026. Google’s Nano Banana 2 includes image text and translation features.

This published example shows a sign being changed for another language. For a designer, that could help when trying different poster versions.

But changing the language can also change the length of the text, the line breaks, and how the page feels.

So I would check more than whether letters appeared. I’d read every word, check the meaning, and look at the spacing.

In a station scene, the same issue applies to signs, tickets, and notices.

Typography has both language requirements and visual requirements. A sign can spell the words correctly but use poor spacing. It can also look balanced while changing an important word. For a poster, separate text accuracy, line breaks, alignment, and hierarchy. Generating lettering inside an image can help develop a concept. For a final layout that needs exact wording, editable text remains useful because a correction does not require generating the entire image again.

2026: Keeping characters recognizable

Another useful development is keeping characters recognizable across a sequence. This is Google’s published storyboard example.

Look at the recurring characters as their poses and positions change. That’s closer to what we need for a story than making one attractive image.

We need enough variation for different shots while keeping the character recognizable.

For the character, I’d compare the shape of the coat, the hair, and the suitcase. Google reports improved consistency, but we should still inspect each shot.

A character can look similar at first glance while a small change becomes distracting when we cut between images.

Subject consistency is also different from pixel identity. A character viewed from behind should not have the same pixels as a front view. The goal is to preserve the features that identify the subject while allowing a physically and visually plausible change of view. In a storyboard, compare costume construction, body proportions, and prop ownership. A repeated face alone is not enough to make the whole sequence continuous.

September 2026: GPT Image 2.5

The official OpenAI documentation lists September eighth, twenty twenty-six snapshots of GPT Image two point five Sunburst and Flare. Sunburst is described as the most capable image model, while Flare is aimed at faster everyday image work. Both are image generation and editing models.

For a design comparison, use the same task and reference images. A product label about capability does not tell us whether it will preserve a particular logo, maintain a character across shots, or reproduce exact lettering in our layout. Those require output-level checks.

Speed also matters differently at different stages. Fast alternatives can help during composition selection. A slower result may be worthwhile for a demanding final edit, but only if the difference is visible and useful. The public model descriptions do not establish an independent ranking or reveal every internal architectural choice.

2026: Give the video model examples

Video tools are also accepting more kinds of reference. Seedance 2.0, announced in February 2026, can use text, images, sound, and video as inputs.

It can generate short video with sound.

Suppose we like the traveler’s appearance in one picture, and we have a separate clip showing the camera movement we want.

Those references communicate different parts of the scene. This could be useful when movement is difficult to describe in words.

We would still need to check the result. Did it follow the movement, and did that movement create the feeling we wanted?

Different references specify different variables. A character image can describe appearance. A motion clip can describe timing and camera behavior. Audio can describe rhythm or an event sequence. Combining references is useful when these roles are clear. Conflicting references still need a decision: a slow camera reference and a rapid musical beat may suggest different editing choices. The model cannot determine the intended relationship from the existence of the files alone.

2026: Edit a video by describing a change

Another change is being able to describe an edit to an existing video. Google introduced Gemini Omni Flash in public preview in June 2026.

Its examples include changing lighting and replacing objects through spoken or written instructions. For this image, we might say, “Keep the traveler.

Make the station lights warmer.” Then we need to watch the whole result. Did the face change? Did an object disappear? Did the sound stay right?

An edit that works in one frame may cause problems elsewhere.

Also, available controls can differ between a demonstration and the version of a tool you can access.

Video editing introduces a preservation problem. The request specifies what should change, but the rest of the clip also matters. Relighting should be checked across moving surfaces and shadows, not only at the first frame. Object replacement should preserve occlusion when something passes in front of it. These are concrete tests of edit behavior. They are more useful than assuming that a fluent natural-language instruction guarantees a localized change.

Video review: frame, motion, sound

Video quality needs tests at more than one timescale. At the frame level, inspect object count, anatomy, lettering, and composition. These are similar to still-image checks. At the sequence level, inspect whether objects persist and whether their motion remains continuous.

Camera motion is a useful stress test for geometry. A wall can look plausible in one frame and bend as the viewpoint changes. Occlusion is another test: an object passing behind a foreground shape should reappear with a consistent identity and position.

Sound adds event timing. A visible impact and its sound should have a deliberate relationship. Dialogue needs a relationship between speech timing and mouth movement. The goal is not simply to have an audio track, but to make the audiovisual events agree with the intended scene.

At an edit, screen direction and prop position affect continuity. An object moving to the right in one shot and left in the next can imply a reversal. That may be intentional, but it should be a choice supported by the spatial setup rather than an accidental generation change.

2026: Video, sound, and keyframe control

FLUX 3 illustrates the move from separate image and sound generation toward joint audiovisual generation. Black Forest Labs released an initial version of FLUX 3 Video on August fourth, twenty twenty-six. Its announcement describes clips up to twenty seconds with audio, image inputs, keyframes, and continuation.

Keyframes specify important visible moments, while the model generates the transition between them. They can provide more concrete temporal control than a style adjective. They still leave intermediate motion to be generated, so a correct first and last frame do not establish that the movement between them is correct.

The same company's August upscaling announcement provides another useful example of a tradeoff: a more creative repair mode can change identity. Higher resolution and stronger reconstruction are therefore separate choices from preservation. These are developer descriptions; an independent comparison needs matched inputs and recorded outputs.

GPT-6 Astra: Help build an interactive scene

OpenAI's GPT-six Astra is a general assistant model for reasoning, coding, and work with tools. It can accept text and image inputs. In a design workflow, the relevant example is building an interactive artifact from a specification, then inspecting and revising the result.

An assistant might write the code for a light transition, inspect a screenshot, and change the timing after feedback. The image-generation tool used within that workflow can be a separate model. It is useful to distinguish the assistant that coordinates the work from the specialized tool that produces an image.

This is also different from a world model predicting the next view, or a VLA producing robot commands. Those systems have different outputs and different tests. The next example makes the assistant's role concrete by turning an interaction request into observable behavior.

Astra: turn a design instruction into a test

An interaction can be specified as an input, a state change, and an output over time. Here the input is a click on a prop. The intended output is a light fade lasting two seconds. The camera and character should remain fixed while that change occurs.

Astra can help write and revise the code for that behavior using tools. The important test is the built artifact. Does one click start one transition? Does the final light level match the instruction? Does a second click restart the transition, reverse it, or do nothing?

Those alternatives are design decisions about state. They should be stated explicitly. Otherwise, a generated implementation can look correct in a screenshot while behaving unpredictably during repeated use. A static image cannot show all the states of an interaction.

The assistant's role is to help construct and inspect the implementation. The test still needs observable input and output. This separates a convincing explanation of an interaction from a working interaction, and connects language-based assistance to a concrete design workflow.

World models: what is being predicted?

A world model predicts how an environment changes, often conditioned on an action. The representation can differ. One model predicts a compact state, another generates future video, and another works with an explicit three-dimensional scene that can be rendered from different viewpoints.

CMU's world-model lecture separates these approaches because they support different operations. A learned latent state can be useful for control without producing a picture. An interactive video model produces visible views but may not expose editable three-dimensional geometry. A scene representation can support rendering, while its physical behavior still needs a model.

The term world model therefore does not name one fixed architecture. The practical question is what state it represents, what action it accepts, and what it predicts. A plausible image is evidence about appearance, not automatically evidence of accurate distance, dynamics, or a persistent environment.

Astra: A picture that responds to movement

This Astra is the world-model research project presented at ICLR 2026. Its first preprint appeared in December 2025. It’s separate from OpenAI’s GPT-6 Astra.

Look at the green-bordered pictures on the left. Those are starting images. The later columns show generated views, with movement controls marked on them.

The model uses earlier observations and action inputs to predict the next part of the video.

That connects to our earlier discussion of conditions: now a movement command helps guide generation. For an interactive scene, we care about both response and consistency.

Does the view change when we ask, and does the place still make sense afterward?

An action-conditioned video model receives information about intended movement, along with earlier observations. That input changes the prediction task: it must produce a plausible continuation that also responds to the action. A prerecorded video only needs to play its next frame. An interactive model must handle different possible actions from the same current view. Evaluation therefore needs response tests as well as visual tests, including repeated actions and movement back toward an earlier view.

World models: test a round trip

A round trip tests whether a generated environment stays consistent over time. Begin outside a doorway, enter, turn away, turn back, and leave again. The sequence revisits earlier geometry rather than continually generating new views that never need to agree with the past.

A model can generate a plausible next frame while gradually changing the doorway's size or location. If its own generated frames become later inputs, small errors can accumulate. Short-term realism and long-term consistency are therefore different evaluation targets.

Object persistence is related. A chair that moves outside the camera view should not automatically cease to exist. Turning back provides a test of what the model retains. An explicit three-dimensional scene, a video history, and a learned hidden state can support persistence in different ways.

This test does not establish all aspects of physics. It specifically checks spatial and temporal agreement under a short sequence of actions. Additional tests would be needed for contact, material behavior, or control accuracy. The test should match the capability being claimed.

VLA: See the scene, read the task, act

VLA stands for vision-language-action. The model receives visual information and an instruction, often along with information about the robot’s current position.

It produces actions for the robot, such as moving an arm or closing a gripper. Gemini Robotics 2, announced in July 2026, is one example.

It includes whole-body control, so movement can involve more than an arm at a table.

For an installation, we might imagine a robot moving props as visitors interact. That would require a complete working system.

The model’s action output is one part; observing the result and dealing with mistakes also matter.

Robot actions can be represented in several ways, such as joint targets, end-effector movement, or a sequence of future controls. The model's output must match the robot interface. A language instruction saying pick up the cup is much less specific than those low-level actions. A complete system needs observations, a policy, motor execution, and updated feedback. The VLA is the learned link between visual-language information and the action representation used by that system.

Robot actions need a coordinate system

An image location and a robot movement use different coordinate systems. A prop appearing forty pixels left of the image center does not directly specify how many centimeters a robot hand should move. The relationship depends on camera geometry, depth, and the robot's current configuration.

A command also needs a reference frame. Left can mean the camera's left, the robot's left, or a direction in the workspace. A movement can be expressed as a change in joint angles or as a target for the end effector. The controller must interpret the representation correctly.

A VLA learns a mapping from observations and language into actions, often with robot-state information. The surrounding system still determines which action representation is used and how the motor commands are executed. A model that predicts the right-looking motion is not by itself a calibrated physical installation.

After movement, new observations close the feedback loop. For a prop-handling installation, the stopping condition might be reaching a marked location or achieving a stable grasp. Defining the frame, distance, feedback, and stop condition turns a vague movement request into a testable task.

Predicting an action’s result and taking action

These systems have different jobs. An assistant might write code for the suitcase interaction. A world model might predict what would happen if it moved.

A VLA might produce commands to move a real suitcase. A system could combine those jobs, then look again after acting.

But a VLA doesn’t necessarily contain a separate world model. And generating a video of an action doesn’t mean a robot has actually performed it.

A planner may ask a world model to predict several candidate outcomes before choosing an action. That is one possible arrangement. Another policy can map observations directly to action commands without generating future images. Neither architecture is proved by watching a successful demonstration. We need a system description to know which components are present, and a task test to know whether their combination works reliably.

Discussion: Design an experience that responds

Let’s turn that into a design choice. Imagine a visitor moves the red suitcase in a station scene. What should the experience do?

Explain something, generate a changed scene, or move a real prop? Choose one with your partner.

Say what the visitor does and what they should notice in response. Include what happens if the system misunderstands them.

For an installation, specify four things: the visitor input, the visible response, the response time, and the recovery behavior. A click that changes light is different from a camera movement that generates a new view, and both differ from moving a physical prop. The input and output determine which technology is needed. This is the point where interaction design becomes a concrete system requirement.

Which setting might help?

If the traveler is too large, try a wider-view instruction or a layout reference.

If the colors feel too strong, guidance may be worth testing where the model supports it.

If one small area is wrong, a local edit may be enough. These are starting points for a test. They aren’t guaranteed fixes.

Describe the visible problem first.

That makes it easier to choose a change that could actually address it, rather than changing everything and hoping the next image works.

The location of an error helps select a control. A wrong global viewpoint may call for a layout reference. A correct scene with one broken hand may call for a local region edit. A repeated style across a large collection may justify a trained adapter. These choices differ in scope and cost. Starting with the smallest relevant intervention makes it easier to preserve the parts that already work.

How to make a fair comparison

Here is a simple comparison we could run. Our question is whether guidance changes subject clarity and variety.

We keep the model, prompt, size, sampler, and steps fixed. We try different guidance values using the same set of seeds at each value.

Then we judge the results using questions we chose beforehand. That last part matters.

If we decide what counts as success only after seeing the images, it’s easy to favor the result we happened to like.

A clear test helps us explain our choice to someone else.

Write the comparison as a small table before running it. The rows can be seeds and the columns guidance values. Every cell gets the same prompt and model. Record failures as well as successful images, because omitting failures changes what the grid represents. If image generation fails for a cell, mark it as missing and rerun that condition rather than silently replacing it with a different seed.

Content, form, and variation

Image evaluation can separate content, form, variation, and purpose. Content includes object count, attributes, and relationships. These can often be checked directly against the request. A hand holding a cup is a different relation from a hand merely appearing near a cup.

Form concerns how the image is organized: silhouette, value structure, edge contrast, and the main focus. These properties connect technical output to art decisions. They can be described specifically even when people prefer different compositions.

Variation concerns the group rather than one image. A set can contain one attractive result and many nearly identical alternatives. Purpose determines which of those properties matter most. A clear instructional poster and an ambiguous dream sequence can reasonably favor different results without making the comparison meaningless.

Discussion 4: Compare the pictures

We’ll spend six minutes on these grids. They come from a published study of guidance using dog images.

This was generation from a category label, rather than a written prompt. Corresponding positions use matched seeds, so compare the same position across the groups.

First, look silently. Then choose two corresponding sets and write down what changes. Look at recognizable features, unwanted details, and variety within each group.

Finally, choose a group for a particular use. Would your choice change for a clear introduction compared with a strange dream scene?

The task is to make a defensible comparison, not to guess the setting with the biggest number. Select corresponding positions, describe shape and color changes, and then inspect variation across each group. A picture can become more recognizable while losing unusual but useful alternatives. That tradeoff is central to generative design: the setting that helps one final image may not be the best setting for exploring possibilities.

Reading a guidance grid

A controlled grid supports two kinds of reading. Across guidance conditions, follow the same seed and describe what changes. Within one guidance condition, compare different seeds and describe the range of outputs. The first comparison focuses on the setting; the second focuses on variation.

For a face, useful observations include outline, eye placement, texture, and exaggerated edges. For a full scene, include camera distance, figure size, and repeated layout patterns. Record what is visible rather than assuming that a stronger setting improved every property.

A grid is strongest when its labels make the comparison unambiguous. Keep model, prompt, and sampling settings beside it. If images have been selected or cropped, state that too. Otherwise, the display can appear more controlled or more diverse than the underlying experiment really was.

Discussion 5: Choose a look for our scene

For this last discussion, choose a camera view and a drawing style for a station scene.

Think back to the wide and close views, and the paper, paint, and ink examples. Take thirty seconds to choose.

Then explain your choice to your partner using two details you can point to in the pictures. Finish by naming one change you would try next.

Keep the change specific enough that you could recognize whether it worked.

Use a formal reason for the choice. A low camera position can increase the apparent scale of a nearby figure. A diagonal edge can connect two regions or create tension. A large quiet area can reserve space for text. These are specific compositional effects. A style label such as cinematic is less useful unless it is translated into decisions about camera, light, color, or editing.

Three questions before we finish

Before we finish, write a short answer to each of these questions. Why can the same prompt give different pictures?

Why might stronger guidance make a picture worse? And what would you save so you could try to repeat a result? Use your own words.

You don’t need an equation. If it helps, explain each answer using the character and station.

A complete answer should name a mechanism and its consequence. For seed, name the changed random state and the resulting variation. For guidance, name the combined predictions and the risk of exaggeration. For reproducibility, name the inputs and model settings that must be recorded. This is a short check that the terms connect to something the system actually does.

Suggested answers

Here are the main answers. Different random starting states can lead to different pictures, even with the same prompt.

Depending on the sampler, randomness can also enter later. Stronger guidance pushes harder along the difference between predictions.

That may make the subject clearer, but it can also create unwanted colors or details and reduce variety.

To repeat a result, save the full setup: the model, prompt, references, seed, and generation settings. Include any adapters.

Exact matching can still depend on software and hardware. Your wording can be different. What matters is whether your explanation connects the setting to what happens.

These answers also explain why a beautiful result is not enough to understand a model. One sample shows an outcome. A matched comparison shows how a controlled change affects outcomes. A saved setup makes that comparison repeatable. Together, these give us a method for discussing generation in concrete terms rather than relying on impressions about a tool's personality or creativity.

A method for the next image

The main controls now have distinct meanings. A prompt and references describe conditions. A seed fixes a random starting process. Guidance combines predictions. A sampler calculates updates. Adapters and fine-tuning change learned behavior. Masks and compositing limit where an edit appears.

The art decisions are just as concrete. Camera position changes perspective. Overlap and converging lines provide depth cues. Hue and lightness affect separation. Edge treatment controls which shapes remain clear. Character and motion continuity connect individual images into a sequence.

A useful experiment changes a relevant variable, preserves the other conditions, and records the results. That method applies to an image edit, a video comparison, or an interactive system. The following source slides point to the university lectures, papers, and official announcements used in this lesson.

University lectures behind this lesson

The technical structure draws on CMU's Generative AI course, including diffusion, text-to-image generation, parameter-efficient adaptation, and world models. MIT's flow and diffusion course provides a clear connection between training targets and numerical sampling. The small numerical examples in this lecture were written for this class.

Stanford's generative-model lectures provide additional technical context. Its graphics and photography materials support the discussion of color, cameras, and composition. These established art and imaging principles are presented alongside recent tools because they explain decisions that remain useful when the product names change.

Papers, demonstrations, and further reading

The primary papers explain the mechanisms behind diffusion, latent representations, guidance, LoRA, image-to-image editing, and spatial conditioning. Related methods such as IP-Adapter and DreamBooth show different ways to use reference information. Their publication dates are kept distinct from the twenty twenty-six product updates.

The recent section uses official model documentation and developer announcements. Those sources establish what was announced and how the developer describes it. They are not independent proof that a model is best for every task. A practical comparison still needs a defined task, matched inputs, and the actual outputs.