```
%pip install -q "openvino>=2023.1.0"
%pip install -q --extra-index-url https://download.pytorch.org/whl/cpu
"diffusers[torch]>=0.9.0"
%pip install -q "huggingface-hub>=0.9.1"
%pip install -q gradio
%pip install -q transformers

━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 38.1/38.1 MB 11.9 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.0/2.0 MB 9.4 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 297.4/297.4 kB 24.6 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 17.1/17.1 MB 45.4 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 91.9/91.9 kB 12.5 MB/s eta
0:00:00
etadata (setup.py) ... ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
313.6/313.6 kB 37.1 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 75.6/75.6 kB 10.8 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 144.8/144.8 kB 19.6 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 8.9/8.9 MB 63.3 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 60.8/60.8 kB 8.5 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 129.9/129.9 kB 17.4 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 77.9/77.9 kB 11.7 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 58.3/58.3 kB 8.4 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 71.9/71.9 kB 10.1 MB/s eta
0:00:00
py (setup.py) ...

from diffusers import StableDiffusionPipeline
import gc

pipe =
StableDiffusionPipeline.from_pretrained("prompthero/openjourney").to("
cpu")
text_encoder = pipe.text_encoder
text_encoder.eval()
unet = pipe.unet
unet.eval()
vae = pipe.vae
vae.eval()
```

```python
del pipe
gc.collect()
```

```
The cache for model files in Transformers v4.22.0 has been updated.
Migrating your old cache. This is a one-time only operation. You can
interrupt this and resume the migration later on by calling
`transformers.utils.move_cache()`.
```

```
{"model_id":"59f7c0b2e4a94a2e8177d2ffd352fc3a","version_major":2,"version_minor":0}
```

```
/usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/
_token.py:88: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your
settings tab (https://huggingface.co/settings/tokens), set it as
secret in your Google Colab and restart your session.
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to
access public models or datasets.
  warnings.warn(
```

```
{"model_id":"3356118024a84ad097a592f28bafdad7","version_major":2,"version_minor":0}

{"model_id":"1f8695b64e2b4f86923662e3c3bdfe62","version_major":2,"version_minor":0}

{"model_id":"0f520a5495b34c81a69b4d349f851525","version_major":2,"version_minor":0}

{"model_id":"9623b159cabc4865bc03d06e53b81901","version_major":2,"version_minor":0}

{"model_id":"2675076e04914fc6b61536e1394a391c","version_major":2,"version_minor":0}

{"model_id":"72993be26116471b8e2e36570b1eb529","version_major":2,"version_minor":0}

{"model_id":"5c560295e32f4085abd3fd2c71b1f8e1","version_major":2,"version_minor":0}

{"model_id":"c279145282444960b82a06e64da8cffe","version_major":2,"version_minor":0}

{"model_id":"a6df6103a53f4dd2b6590f1374221317","version_major":2,"version_minor":0}

{"model_id":"252bfb50ad6c49adabd3f9f410c36244","version_major":2,"version_minor":0}
```

```
{"model_id":"c6304adfaac64e5e871c4ca796f0af1d","version_major":2,"version_minor":0}

{"model_id":"324273adf6244866b7bb1d1763eb9f95","version_major":2,"version_minor":0}

{"model_id":"041d47da26b241e9a6f9615a943b9cad","version_major":2,"version_minor":0}

{"model_id":"ccc20beef04f4a328de754fb24872145","version_major":2,"version_minor":0}

{"model_id":"d93d20dc713c45c88ff1419985c20f80","version_major":2,"version_minor":0}

{"model_id":"470d01cc21714a5385a43655b2c713a5","version_major":2,"version_minor":0}

{"model_id":"b978d4b11672478a83fc5575a74a1437","version_major":2,"version_minor":0}

{"model_id":"4c2750ad53e64921ada4f66e1a55a9c1","version_major":2,"version_minor":0}
```

```
29
```

```python
from pathlib import Path
import torch
import openvino as ov

TEXT_ENCODER_OV_PATH = Path("text_encoder.xml")

def cleanup_torchscript_cache():
    """
    Helper for removing cached model representation
    """
    torch._C._jit_clear_class_registry()
    torch.jit._recursive.concrete_type_store = torch.jit._recursive.ConcreteTypeStore()
    torch.jit._state._clear_class_state()

def convert_encoder(text_encoder: torch.nn.Module, ir_path:Path):
    """
    Convert Text Encoder mode.
    Function accepts text encoder model, and prepares example inputs for conversion,
    Parameters:
        text_encoder (torch.nn.Module): text_encoder model from Stable Diffusion pipeline
        ir_path (Path): File for storing model
    Returns:
        None
    """
    input_ids = torch.ones((1, 77), dtype=torch.long)
    # switch model to inference mode
    text_encoder.eval()

    # disable gradients calculation for reducing memory consumption
    with torch.no_grad():
        # Export model to IR format
        ov_model = ov.convert_model(text_encoder,
example_input=input_ids, input=[(1,77),])
    ov.save_model(ov_model, ir_path)
    del ov_model
    cleanup_torchscript_cache()
    print(f'Text Encoder successfully converted to IR and saved to
{ir_path}')


if not TEXT_ENCODER_OV_PATH.exists():
    convert_encoder(text_encoder, TEXT_ENCODER_OV_PATH)
else:
    print(f"Text encoder will be loaded from {TEXT_ENCODER_OV_PATH}")

del text_encoder
gc.collect()
```

```
/usr/local/lib/python3.10/dist-packages/transformers/
modeling_utils.py:4193: FutureWarning:
`_is_quantized_training_enabled` is going to be deprecated in
transformers 4.39.0. Please use `model.hf_quantizer.is_trainable`
instead
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/transformers/modeling_attn_mas
k_utils.py:86: TracerWarning: Converting a tensor to a Python boolean
might cause the trace to be incorrect. We can't record the data flow
of Python values, so this value will be treated as a constant in the
future. This means that the trace might not generalize to other
inputs!
  if input_shape[-1] > 1 or self.sliding_window is not None:
/usr/local/lib/python3.10/dist-packages/transformers/modeling_attn_mas
k_utils.py:162: TracerWarning: Converting a tensor to a Python boolean
might cause the trace to be incorrect. We can't record the data flow
of Python values, so this value will be treated as a constant in the
future. This means that the trace might not generalize to other
inputs!
  if past_key_values_length > 0:
/usr/local/lib/python3.10/dist-packages/transformers/models/clip/model
ing_clip.py:281: TracerWarning: Converting a tensor to a Python
boolean might cause the trace to be incorrect. We can't record the
data flow of Python values, so this value will be treated as a
constant in the future. This means that the trace might not generalize
```

```
to other inputs!
  if attn_weights.size() != (bsz * self.num_heads, tgt_len, src_len):
/usr/local/lib/python3.10/dist-packages/transformers/models/clip/model
ing_clip.py:289: TracerWarning: Converting a tensor to a Python
boolean might cause the trace to be incorrect. We can't record the
data flow of Python values, so this value will be treated as a
constant in the future. This means that the trace might not generalize
to other inputs!
  if causal_attention_mask.size() != (bsz, 1, tgt_len, src_len):
/usr/local/lib/python3.10/dist-packages/transformers/models/clip/model
ing_clip.py:321: TracerWarning: Converting a tensor to a Python
boolean might cause the trace to be incorrect. We can't record the
data flow of Python values, so this value will be treated as a
constant in the future. This means that the trace might not generalize
to other inputs!
  if attn_output.size() != (bsz * self.num_heads, tgt_len,
self.head_dim):
```

```
Text Encoder successfully converted to IR and saved to
text_encoder.xml
```

```
1264
```

```python
import numpy as np

UNET_OV_PATH = Path('unet.xml')

dtype_mapping = {
    torch.float32: ov.Type.f32,
    torch.float64: ov.Type.f64
}


def convert_unet(unet:torch.nn.Module, ir_path:Path):
    """
    Convert U-net model to IR format.
    Function accepts unet model, prepares example inputs for
conversion,
    Parameters:
        unet (StableDiffusionPipeline): unet from Stable Diffusion
pipeline
        ir_path (Path): File for storing model
    Returns:
        None
    """
    # prepare inputs
    encoder_hidden_state = torch.ones((2, 77, 768))
    latents_shape = (2, 4, 512 // 8, 512 // 8)
    latents = torch.randn(latents_shape)
    t = torch.from_numpy(np.array(1, dtype=float))
    dummy_inputs = (latents, t, encoder_hidden_state)
    input_info = []
    for input_tensor in dummy_inputs:
        shape = ov.PartialShape(tuple(input_tensor.shape))
        element_type = dtype_mapping[input_tensor.dtype]
        input_info.append((shape, element_type))

    unet.eval()
    with torch.no_grad():
        ov_model = ov.convert_model(unet, example_input=dummy_inputs,
input=input_info)
    ov.save_model(ov_model, ir_path)
    del ov_model
    cleanup_torchscript_cache()
    print(f'Unet successfully converted to IR and saved to {ir_path}')


if not UNET_OV_PATH.exists():
    convert_unet(unet, UNET_OV_PATH)
    gc.collect()
else:
    print(f"Unet will be loaded from {UNET_OV_PATH}")
del unet
gc.collect()
```

```
/usr/local/lib/python3.10/dist-packages/diffusers/models/unets/
unet_2d_condition.py:1110: TracerWarning: Converting a tensor to a
Python boolean might cause the trace to be incorrect. We can't record
the data flow of Python values, so this value will be treated as a
constant in the future. This means that the trace might not generalize
to other inputs!
  if dim % default_overall_up_factor != 0:
/usr/local/lib/python3.10/dist-packages/diffusers/models/downsampling.
py:137: TracerWarning: Converting a tensor to a Python boolean might
cause the trace to be incorrect. We can't record the data flow of
Python values, so this value will be treated as a constant in the
future. This means that the trace might not generalize to other
inputs!
  assert hidden_states.shape[1] == self.channels
/usr/local/lib/python3.10/dist-packages/diffusers/models/downsampling.
py:146: TracerWarning: Converting a tensor to a Python boolean might
cause the trace to be incorrect. We can't record the data flow of
Python values, so this value will be treated as a constant in the
future. This means that the trace might not generalize to other
inputs!
  assert hidden_states.shape[1] == self.channels
/usr/local/lib/python3.10/dist-packages/diffusers/models/upsampling.py
:149: TracerWarning: Converting a tensor to a Python boolean might
cause the trace to be incorrect. We can't record the data flow of
Python values, so this value will be treated as a constant in the
```

```
future. This means that the trace might not generalize to other
inputs!
  assert hidden_states.shape[1] == self.channels
/usr/local/lib/python3.10/dist-packages/diffusers/models/upsampling.py
:165: TracerWarning: Converting a tensor to a Python boolean might
cause the trace to be incorrect. We can't record the data flow of
Python values, so this value will be treated as a constant in the
future. This means that the trace might not generalize to other
inputs!
  if hidden_states.shape[0] >= 64:

Unet successfully converted to IR and saved to unet.xml

0
```

```python
VAE_ENCODER_OV_PATH = Path("vae_encoder.xml")

def convert_vae_encoder(vae: torch.nn.Module, ir_path: Path):
    """
    Convert VAE model for encoding to IR format.
    Function accepts vae model, creates wrapper class for export only
necessary for inference part,
    prepares example inputs for conversion,
    Parameters:
        vae (torch.nn.Module): VAE model from StableDiffusio pipeline
        ir_path (Path): File for storing model
    Returns:
        None
    """
    class VAEEncoderWrapper(torch.nn.Module):
        def __init__(self, vae):
            super().__init__()
            self.vae = vae

        def forward(self, image):
            return self.vae.encode(x=image)["latent_dist"].sample()
    vae_encoder = VAEEncoderWrapper(vae)
    vae_encoder.eval()
    image = torch.zeros((1, 3, 512, 512))
    with torch.no_grad():
        ov_model = ov.convert_model(vae_encoder, example_input=image,
input=[((1,3,512,512),)])
    ov.save_model(ov_model, ir_path)
    del ov_model
    cleanup_torchscript_cache()
    print(f'VAE encoder successfully converted to IR and saved to
{ir_path}')

if not VAE_ENCODER_OV_PATH.exists():
    convert_vae_encoder(vae, VAE_ENCODER_OV_PATH)
```

```
else:
    print(f"VAE encoder will be loaded from {VAE_ENCODER_OV_PATH}")

VAE_DECODER_OV_PATH = Path('vae_decoder.xml')

def convert_vae_decoder(vae: torch.nn.Module, ir_path: Path):
    """
    Convert VAE model for decoding to IR format.
    Function accepts vae model, creates wrapper class for export only
necessary for inference part,
    prepares example inputs for conversion,
    Parameters:
        vae (torch.nn.Module): VAE model frm StableDiffusion pipeline
        ir_path (Path): File for storing model
    Returns:
        None
    """
    class VAEDecoderWrapper(torch.nn.Module):
        def __init__(self, vae):
            super().__init__()
            self.vae = vae

        def forward(self, latents):
            return self.vae.decode(latents)

    vae_decoder = VAEDecoderWrapper(vae)
    latents = torch.zeros((1, 4, 64, 64))

    vae_decoder.eval()
    with torch.no_grad():
        ov_model = ov.convert_model(vae_decoder,
example_input=latents, input=[((1,4,64,64),)])
    ov.save_model(ov_model, ir_path)
    del ov_model
    cleanup_torchscript_cache()
    print(f'VAE decoder successfully converted to IR and saved to
{ir_path}')


if not VAE_DECODER_OV_PATH.exists():
    convert_vae_decoder(vae, VAE_DECODER_OV_PATH)
else:
    print(f"VAE decoder will be loaded from {VAE_DECODER_OV_PATH}")

del vae
gc.collect()
```

```
/usr/local/lib/python3.10/dist-packages/torch/jit/_trace.py:1102:
TracerWarning: Trace had nondeterministic nodes. Did you forget
call .eval() on your model? Nodes:
     %2494 : Float(1, 4, 64, 64, strides=[16384, 4096, 64, 1],
requires_grad=0, device=cpu) = aten::randn(%2488, %2489, %2490, %2491,
%2492, %2493) #
/usr/local/lib/python3.10/dist-packages/diffusers/utils/torch_utils.py
:80:0
This may cause errors in trace checking. To disable trace checking,
pass check_trace=False to torch.jit.trace()
  _check_trace(
/usr/local/lib/python3.10/dist-packages/torch/jit/_trace.py:1102:
TracerWarning: Output nr 1. of the traced function does not match the
corresponding output of the Python function. Detailed error:
Tensor-likes are not close!

Mismatched elements: 10318 / 16384 (63.0%)
Greatest absolute difference: 0.0020890235900878906 at index (0, 2,
63, 63) (up to 1e-05 allowed)
Greatest relative difference: 0.00297151261570422 at index (0, 3, 2,
62) (up to 1e-05 allowed)
  _check_trace(

VAE encoder successfully converted to IR and saved to vae_encoder.xml
VAE decoder successfully converted to IR and saved to vae_decoder.xml

4074
```

```python
import inspect
from typing import List, Optional, Union, Dict
import numpy as np
import PIL
import cv2

from transformers import CLIPTokenizer
from diffusers.pipelines.pipeline_utils import DiffusionPipeline
from diffusers.schedulers import DDIMScheduler, LMSDiscreteScheduler,
PNDMScheduler
from openvino.runtime import Model


def scale_fit_to_window(dst_width:int, dst_height:int,
image_width:int, image_height:int):
    """
    Preprocessing helper function for calculating image size for
resize with peserving original aspect ratio
    and fitting image to specific window size

    Parameters:
      dst_width (int): destination window width
      dst_height (int): destination window height
      image_width (int): source image width
      image_height (int): source image height
    Returns:
      result_width (int): calculated width for resize
      result_height (int): calculated height for resize
    """
    im_scale = min(dst_height / image_height, dst_width / image_width)
    return int(im_scale * image_width), int(im_scale * image_height)


def preprocess(image: PIL.Image.Image):
    """
    Image preprocessing function. Takes image in PIL.Image format,
resizes it to keep aspect ration and fits to model input window
512x512,
    then converts it to np.ndarray and adds padding with zeros on
right or bottom side of image (depends from aspect ratio), after that
    converts data to float32 data type and change range of values from
[0, 255] to [-1, 1], finally, converts data layout from planar NHWC to
NCHW.
    The function returns preprocessed input tensor and padding size,
which can be used in postprocessing.

    Parameters:
      image (PIL.Image.Image): input image
    Returns:
       image (np.ndarray): preprocessed image tensor
       meta (Dict): dictionary with preprocessing metadata info
    """
    src_width, src_height = image.size
    dst_width, dst_height = scale_fit_to_window(
        512, 512, src_width, src_height)
    image = np.array(image.resize((dst_width, dst_height),
                     resample=PIL.Image.Resampling.LANCZOS))[None, :]
    pad_width = 512 - dst_width
    pad_height = 512 - dst_height
    pad = ((0, 0), (0, pad_height), (0, pad_width), (0, 0))
    image = np.pad(image, pad, mode="constant")
    image = image.astype(np.float32) / 255.0
    image = 2.0 * image - 1.0
    image = image.transpose(0, 3, 1, 2)
    return image, {"padding": pad, "src_width": src_width,
"src_height": src_height}


class OVStableDiffusionPipeline(DiffusionPipeline):
    def __init__(
        self,
        vae_decoder: Model,
        text_encoder: Model,
        tokenizer: CLIPTokenizer,
        unet: Model,
        scheduler: Union[DDIMScheduler, PNDMScheduler,
LMSDiscreteScheduler],
        vae_encoder: Model = None,
    ):
        """
        Pipeline for text-to-image generation using Stable Diffusion.
        Parameters:
            vae (Model):
                Variational Auto-Encoder (VAE) Model to decode images
to and from latent representations.
            text_encoder (Model):
                Frozen text-encoder. Stable Diffusion uses the text
portion of

[CLIP](https://huggingface.co/docs/transformers/model_doc/clip#transfo
rmers.CLIPTextModel), specifically
                the
clip-vit-large-patch14(https://huggingface.co/openai/clip-vit-large-
patch14) variant.
            tokenizer (CLIPTokenizer):
                Tokenizer of class
CLIPTokenizer(https://huggingface.co/docs/transformers/v4.21.0/en/
model_doc/clip#transformers.CLIPTokenizer).
            unet (Model): Conditional U-Net architecture to denoise
the encoded image latents.
            scheduler (SchedulerMixin):
                A scheduler to be used in combination with unet to
denoise the encoded image latents. Can be one of
                DDIMScheduler, LMSDiscreteScheduler, or PNDMScheduler.
        """
        super().__init__()
        self.scheduler = scheduler
        self.vae_decoder = vae_decoder
        self.vae_encoder = vae_encoder
        self.text_encoder = text_encoder
        self.unet = unet
        self._text_encoder_output = text_encoder.output(0)
        self._unet_output = unet.output(0)
        self._vae_d_output = vae_decoder.output(0)
        self._vae_e_output = vae_encoder.output(0) if vae_encoder is
not None else None
        self.height = 512
        self.width = 512
        self.tokenizer = tokenizer

    def __call__(
        self,
        prompt: Union[str, List[str]],
        image: PIL.Image.Image = None,
        num_inference_steps: Optional[int] = 50,
        negative_prompt: Union[str, List[str]] = None,
        guidance_scale: Optional[float] = 7.5,
        eta: Optional[float] = 0.0,
        output_type: Optional[str] = "pil",
        seed: Optional[int] = None,
        strength: float = 1.0,
        gif: Optional[bool] = False,
        **kwargs,
    ):
        """
        Function invoked when calling the pipeline for generation.
        Parameters:
            prompt (str or List[str]):
                The prompt or prompts to guide the image generation.
            image (PIL.Image.Image, *optional*, None):
                 Intinal image for generation.
            num_inference_steps (int, *optional*, defaults to 50):
                The number of denoising steps. More denoising steps
usually lead to a higher quality image at the
                expense of slower inference.
            negative_prompt (str or List[str]):
                The negative prompt or prompts to guide the image
generation.
            guidance_scale (float, *optional*, defaults to 7.5):
                Guidance scale as defined in Classifier-Free Diffusion
Guidance(https://arxiv.org/abs/2207.12598).
                guidance_scale is defined as `w` of equation 2.
                Higher guidance scale encourages to generate images
that are closely linked to the text prompt,
                usually at the expense of lower image quality.
            eta (float, *optional*, defaults to 0.0):
                Corresponds to parameter eta (η) in the DDIM paper:
https://arxiv.org/abs/2010.02502. Only applies to
                [DDIMScheduler], will be ignored for others.
            output_type (`str`, *optional*, defaults to "pil"):
                The output format of the generate image. Choose
between
                [PIL](https://pillow.readthedocs.io/en/stable/):
PIL.Image.Image or np.array.
            seed (int, *optional*, None):
                Seed for random generator state initialization.
            gif (bool, *optional*, False):
                Flag for storing all steps results or not.
        Returns:
            Dictionary with keys:
                sample - the last generated image PIL.Image.Image or
np.array
                iterations - *optional* (if gif=True) images for all
```

```
diffusion steps, List of PIL.Image.Image or np.array.
        """
        if seed is not None:
            np.random.seed(seed)

        img_buffer = []
        do_classifier_free_guidance = guidance_scale > 1.0
        # get prompt text embeddings
        text_embeddings = self._encode_prompt(prompt,
do_classifier_free_guidance=do_classifier_free_guidance,
negative_prompt=negative_prompt)

        # set timesteps
        accepts_offset = "offset" in
set(inspect.signature(self.scheduler.set_timesteps).parameters.keys())
        extra_set_kwargs = {}
        if accepts_offset:
            extra_set_kwargs["offset"] = 1

        self.scheduler.set_timesteps(num_inference_steps,
**extra_set_kwargs)
        timesteps, num_inference_steps =
self.get_timesteps(num_inference_steps, strength)
        latent_timestep = timesteps[:1]

        # get the initial random noise unless the user supplied it
        latents, meta = self.prepare_latents(image, latent_timestep)

        # prepare extra kwargs for the scheduler step, since not all
schedulers have the same signature
        # eta (η) is only used with the DDIMScheduler, it will be
ignored for other schedulers.
        # eta corresponds to η in DDIM paper:
https://arxiv.org/abs/2010.02502
        # and should be between [0, 1]
        accepts_eta = "eta" in
set(inspect.signature(self.scheduler.step).parameters.keys())
        extra_step_kwargs = {}
        if accepts_eta:
            extra_step_kwargs["eta"] = eta

        for i, t in enumerate(self.progress_bar(timesteps)):
            # expand the latents if you are doing classifier free
guidance
            latent_model_input = np.concatenate([latents] * 2) if
do_classifier_free_guidance else latents
            latent_model_input =
self.scheduler.scale_model_input(latent_model_input, t)

            # predict the noise residual
            noise_pred = self.unet([latent_model_input, t,
text_embeddings])[self._unet_output]
            # perform guidance
            if do_classifier_free_guidance:
                noise_pred_uncond, noise_pred_text = noise_pred[0],
noise_pred[1]
                noise_pred = noise_pred_uncond + guidance_scale *
(noise_pred_text - noise_pred_uncond)

            # compute the previous noisy sample x_t -> x_t-1
            latents =
self.scheduler.step(torch.from_numpy(noise_pred), t,
torch.from_numpy(latents), **extra_step_kwargs)["prev_sample"].numpy()
            if gif:
                image = self.vae_decoder(latents * (1 / 0.18215))
[self._vae_d_output]
                image = self.postprocess_image(image, meta,
output_type)
                img_buffer.extend(image)

        # scale and decode the image latents with vae
        image = self.vae_decoder(latents * (1 / 0.18215))
[self._vae_d_output]

        image = self.postprocess_image(image, meta, output_type)
        return {"sample": image, 'iterations': img_buffer}

    def _encode_prompt(self, prompt:Union[str, List[str]],
num_images_per_prompt:int = 1, do_classifier_free_guidance:bool =
True, negative_prompt:Union[str, List[str]] = None):
        """
        Encodes the prompt into text encoder hidden states.

        Parameters:
            prompt (str or list(str)): prompt to be encoded
            num_images_per_prompt (int): number of images that should
be generated per prompt
            do_classifier_free_guidance (bool): whether to use
classifier free guidance or not
            negative_prompt (str or list(str)): negative prompt to be
encoded
        Returns:
            text_embeddings (np.ndarray): text encoder hidden states
        """
        batch_size = len(prompt) if isinstance(prompt, list) else 1

        # tokenize input prompts
        text_inputs = self.tokenizer(
            prompt,
            padding="max_length",
            max_length=self.tokenizer.model_max_length,
            truncation=True,
            return_tensors="np",
        )
        text_input_ids = text_inputs.input_ids

        text_embeddings = self.text_encoder(
            text_input_ids)[self._text_encoder_output]

        # duplicate text embeddings for each generation per prompt
        if num_images_per_prompt != 1:
            bs_embed, seq_len, _ = text_embeddings.shape
            text_embeddings = np.tile(
                text_embeddings, (1, num_images_per_prompt, 1))
            text_embeddings = np.reshape(
                text_embeddings, (bs_embed * num_images_per_prompt,
seq_len, -1))

        # get unconditional embeddings for classifier free guidance
        if do_classifier_free_guidance:
            uncond_tokens: List[str]
            max_length = text_input_ids.shape[-1]
            if negative_prompt is None:
                uncond_tokens = [""] * batch_size
            elif isinstance(negative_prompt, str):
                uncond_tokens = [negative_prompt]
            else:
                uncond_tokens = negative_prompt
            uncond_input = self.tokenizer(
                uncond_tokens,
                padding="max_length",
                max_length=max_length,
                truncation=True,
                return_tensors="np",
            )

            uncond_embeddings =
self.text_encoder(uncond_input.input_ids)[self._text_encoder_output]

            # duplicate unconditional embeddings for each generation
per prompt, using mps friendly method
            seq_len = uncond_embeddings.shape[1]
            uncond_embeddings = np.tile(uncond_embeddings, (1,
num_images_per_prompt, 1))
            uncond_embeddings = np.reshape(uncond_embeddings,
(batch_size * num_images_per_prompt, seq_len, -1))

            # For classifier free guidance, we need to do two forward
passes.
            # Here we concatenate the unconditional and text
```

```
embeddings into a single batch
            # to avoid doing two forward passes
            text_embeddings = np.concatenate([uncond_embeddings,
text_embeddings])

        return text_embeddings


    def prepare_latents(self, image:PIL.Image.Image = None,
latent_timestep:torch.Tensor = None):
        """
        Function for getting initial latents for starting generation

        Parameters:
            image (PIL.Image.Image, *optional*, None):
                Input image for generation, if not provided randon
noise will be used as starting point
            latent_timestep (torch.Tensor, *optional*, None):
                Predicted by scheduler initial step for image
generation, required for latent image mixing with nosie
        Returns:
            latents (np.ndarray):
                Image encoded in latent space
        """
        latents_shape = (1, 4, self.height // 8, self.width // 8)
        noise = np.random.randn(*latents_shape).astype(np.float32)
        if image is None:
            # if you use LMSDiscreteScheduler, let's make sure latents
are multiplied by sigmas
            if isinstance(self.scheduler, LMSDiscreteScheduler):
                noise = noise * self.scheduler.sigmas[0].numpy()
                return noise, {}
        input_image, meta = preprocess(image)
        latents = self.vae_encoder(input_image)[self._vae_e_output] *
0.18215
        latents = self.scheduler.add_noise(torch.from_numpy(latents),
torch.from_numpy(noise), latent_timestep).numpy()
        return latents, meta

    def postprocess_image(self, image:np.ndarray, meta:Dict,
output_type:str = "pil"):
        """
        Postprocessing for decoded image. Takes generated image
decoded by VAE decoder, unpad it to initila image size (if required),
        normalize and convert to [0, 255] pixels range. Optionally,
convertes it from np.ndarray to PIL.Image format

        Parameters:
            image (np.ndarray):
                Generated image
            meta (Dict):
                Metadata obtained on latents preparing step, can be
empty
            output_type (str, *optional*, pil):
                Output format for result, can be pil or numpy
        Returns:
            image (List of np.ndarray or PIL.Image.Image):
                Postprocessed images
        """
        if "padding" in meta:
            pad = meta["padding"]
            (_, end_h), (_, end_w) = pad[1:3]
            h, w = image.shape[2:]
            unpad_h = h - end_h
            unpad_w = w - end_w
            image = image[:, :, :unpad_h, :unpad_w]
        image = np.clip(image / 2 + 0.5, 0, 1)
        image = np.transpose(image, (0, 2, 3, 1))
        # 9. Convert to PIL
        if output_type == "pil":
            image = self.numpy_to_pil(image)
            if "src_height" in meta:
                orig_height, orig_width = meta["src_height"],
meta["src_width"]
                image = [img.resize((orig_width, orig_height),
                                    PIL.Image.Resampling.LANCZOS) for
img in image]
        else:
            if "src_height" in meta:
                orig_height, orig_width = meta["src_height"],
meta["src_width"]
                image = [cv2.resize(img, (orig_width, orig_width))
                         for img in image]
        return image

    def get_timesteps(self, num_inference_steps:int, strength:float):
        """
        Helper function for getting scheduler timesteps for generation
        In case of image-to-image generation, it updates number of
steps according to strength

        Parameters:
           num_inference_steps (int):
              number of inference steps for generation
           strength (float):
               value between 0.0 and 1.0, that controls the amount of
noise that is added to the input image.
               Values that approach 1.0 enable lots of variations but
will also produce images that are not semantically consistent with the
```

```
input.
        """
        # get the original timestep using init_timestep
        init_timestep = min(int(num_inference_steps * strength),
num_inference_steps)

        t_start = max(num_inference_steps - init_timestep, 0)
        timesteps = self.scheduler.timesteps[t_start:]

        return timesteps, num_inference_steps - t_start

core = ov.Core()

import ipywidgets as widgets


device = widgets.Dropdown(
    options=core.available_devices + ["AUTO"],
    value='CPU',
    description='Device:',
    disabled=False,
)

device

{"model_id":"ed99621118924e56ac11c2b1fd34b17d","version_major":2,"vers
ion_minor":0}

text_enc = core.compile_model(TEXT_ENCODER_OV_PATH, device.value)

unet_model = core.compile_model(UNET_OV_PATH, device.value)

ov_config = {"INFERENCE_PRECISION_HINT": "f32"} if device.value !=
"CPU" else {}

vae_decoder = core.compile_model(VAE_DECODER_OV_PATH, device.value,
ov_config)
vae_encoder = core.compile_model(VAE_ENCODER_OV_PATH, device.value,
ov_config)

from transformers import CLIPTokenizer
from diffusers.schedulers import LMSDiscreteScheduler

lms = LMSDiscreteScheduler(
    beta_start=0.00085,
    beta_end=0.012,
    beta_schedule="scaled_linear"
)
tokenizer = CLIPTokenizer.from_pretrained('openai/clip-vit-large-
patch14')

ov_pipe = OVStableDiffusionPipeline(
    tokenizer=tokenizer,
    text_encoder=text_enc,
    unet=unet_model,
    vae_encoder=vae_encoder,
    vae_decoder=vae_decoder,
    scheduler=lms
)
```

{"model_id":"c6657177a9164f6f8cdbaf69597f056c","version_major":2,"version_minor":0}

{"model_id":"22bcbc9c88524bdc98ddc37ab2847f01","version_major":2,"version_minor":0}

{"model_id":"f875d895bc024203b9d364bb010bdd6f","version_major":2,"version_minor":0}

{"model_id":"bc9ad7b4a5d34a348aa23aa22b079dac","version_major":2,"version_minor":0}

{"model_id":"b6556e4515e44e1e99d4e995aa8342b6","version_major":2,"version_minor":0}

```
import ipywidgets as widgets
sample_text = ('A girl is swimming in a swimming pool.' )
text_prompt = widgets.Text(value=sample_text, description='your text')
num_steps = widgets.IntSlider(min=1, max=50, value=20,
description='steps:')
seed = widgets.IntSlider(min=0, max=10000000, description='seed: ',
value=42)
widgets.VBox([text_prompt, seed, num_steps])
```

{"model_id":"e860d7fad1c14bc1845c6c12df91e3ca","version_major":2,"version_minor":0}

```
print('Pipeline settings')
print(f'Input text: {text_prompt.value}')
print(f'Seed: {seed.value}')
print(f'Number of steps: {num_steps.value}')
```

```
Pipeline settings
Input text: A purple butterfly on a flower.
Seed: 42
Number of steps: 20
```

```
result = ov_pipe(text_prompt.value,
num_inference_steps=num_steps.value, seed=seed.value)
```

{"model_id":"a3d7261ca2a64f71abd0c7e5247f93ce","version_major":2,"version_minor":0}

```
final_image = result['sample'][0]
if result['iterations']:
    all_frames = result['iterations']
    img = next(iter(all_frames))
    img.save(fp='result.gif', format='GIF',
append_images=iter(all_frames), save_all=True,
duration=len(all_frames) * 5, loop=0)
final_image.save('result.png')
```

```
import ipywidgets as widgets
sample_text = ('A red apple on a table' )
text_prompt = widgets.Text(value=sample_text, description='your text')
num_steps = widgets.IntSlider(min=1, max=50, value=20,
description='steps:')
seed = widgets.IntSlider(min=0, max=10000000, description='seed: ',
value=42)
widgets.VBox([text_prompt, seed, num_steps])
```

{"model_id":"9598bdb340d94c9789f96e9489e67acc","version_major":2,"version_minor":0}

```
import ipywidgets as widgets

text = '\n\t'.join(text_prompt.value.split('.'))
print("Input text:")
print("\t" + text)
display(final_image)
```

```
Input text:
	A red apple on a table
```

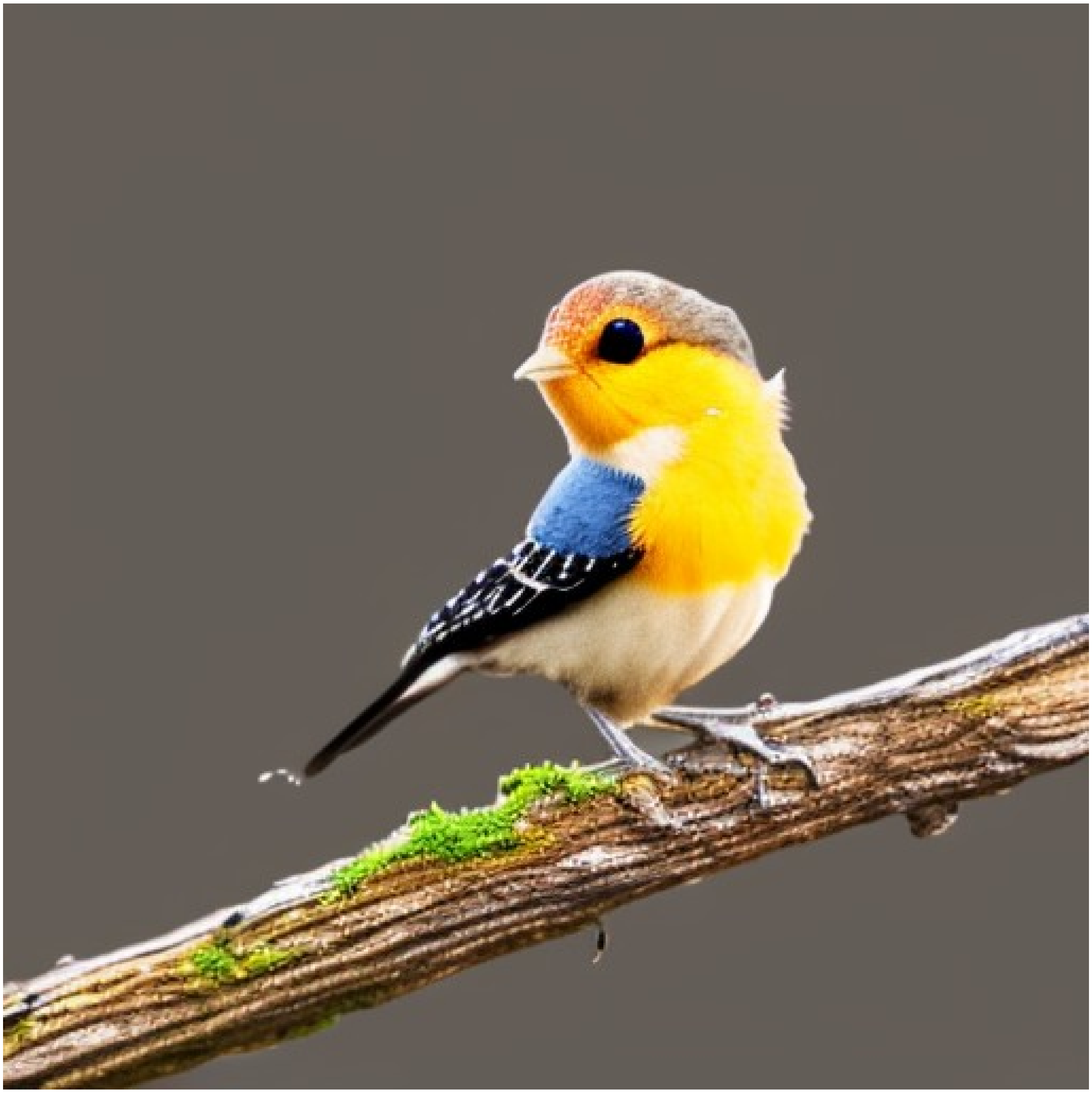
```
import ipywidgets as widgets
sample_text = ('A small bird perched on a branch.' )
text_prompt = widgets.Text(value=sample_text, description='your text')
num_steps = widgets.IntSlider(min=1, max=50, value=20,
description='steps:')
seed = widgets.IntSlider(min=0, max=10000000, description='seed: ',
value=42)
widgets.VBox([text_prompt, seed, num_steps])
```

{"model_id":"a1e0200576ca4cd7a82b7a6eee305e5e","version_major":2,"version_minor":0}

```python
import ipywidgets as widgets

text = '\n\t'.join(text_prompt.value.split('.'))
print("Input text:")
print("\t" + text)
display(final_image)
```

```
Input text:
	A small bird perched on a branch
	
```



```python
import ipywidgets as widgets
sample_text = ('A green frog sitting on a lily pad.' )
text_prompt = widgets.Text(value=sample_text, description='your text')
num_steps = widgets.IntSlider(min=1, max=50, value=20,
description='steps:')
seed = widgets.IntSlider(min=0, max=10000000, description='seed: ',
value=42)
widgets.VBox([text_prompt, seed, num_steps])
```

{"model_id":"3b03ddef942d4f6ba90b8b1dc940fea8","version_major":2,"version_minor":0}

```python
import ipywidgets as widgets

text = '\n\t'.join(text_prompt.value.split('.'))
print("Input text:")
print("\t" + text)
display(final_image)
```

```
Input text:
	A green frog sitting on a lily pad
	
```

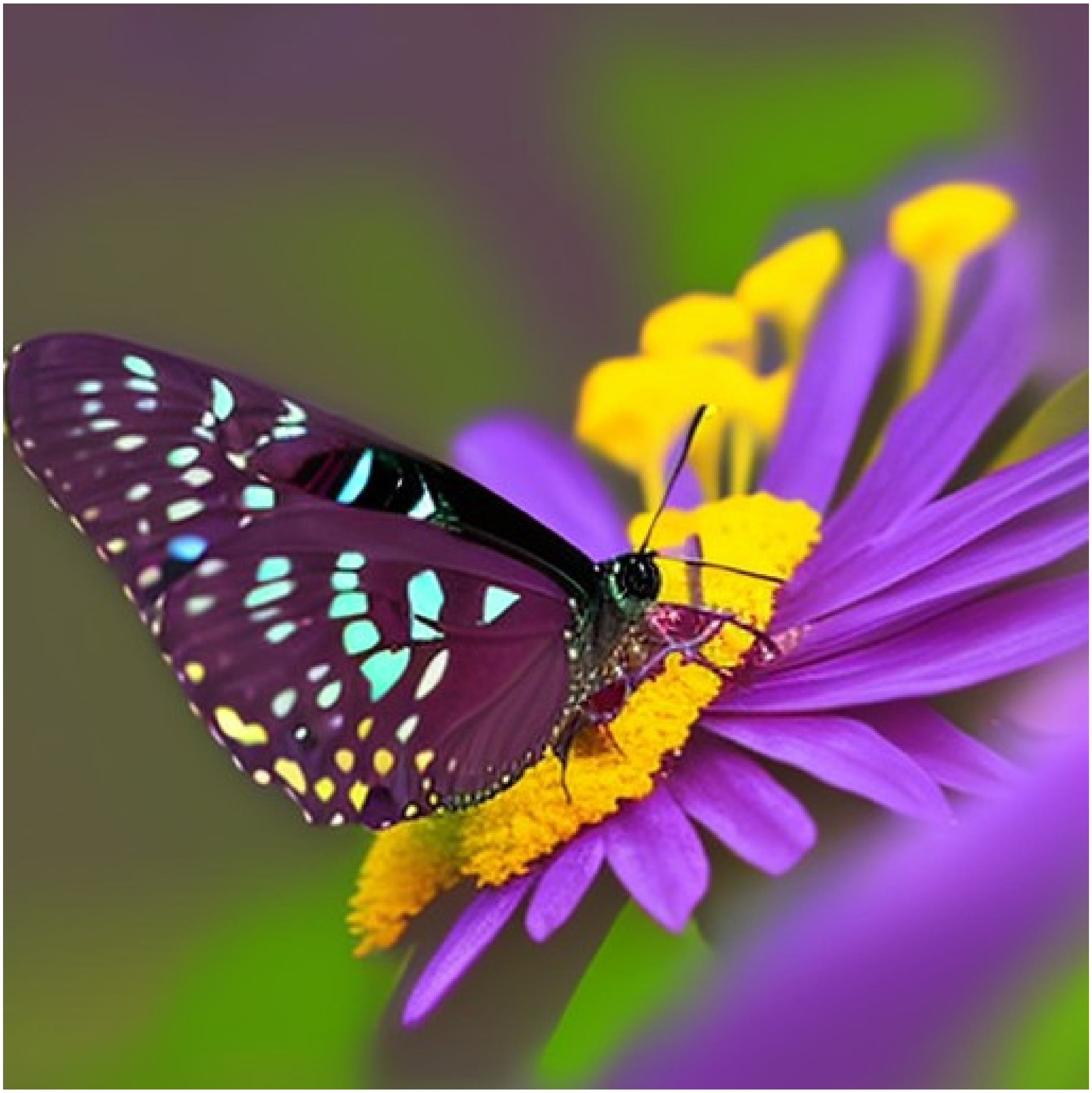
```python
import ipywidgets as widgets
sample_text = ('A purple butterfly on a flower.' )
text_prompt = widgets.Text(value=sample_text, description='your text')
num_steps = widgets.IntSlider(min=1, max=50, value=20,
description='steps:')
seed = widgets.IntSlider(min=0, max=10000000, description='seed: ',
value=42)
widgets.VBox([text_prompt, seed, num_steps])
```

{"model_id":"bfa2a393012d40abb448f4a8f92229b4","version_major":2,"version_minor":0}

```
import ipywidgets as widgets

text = '\n\t'.join(text_prompt.value.split('.'))
print("Input text:")
print("\t" + text)
display(final_image)
```

```
Input text:
	A purple butterfly on a flower
	
```



```
import ipywidgets as widgets
sample_text = ('A brown dog chasing a ball.' )
text_prompt = widgets.Text(value=sample_text, description='your text')
num_steps = widgets.IntSlider(min=1, max=50, value=20,
description='steps:')
seed = widgets.IntSlider(min=0, max=10000000, description='seed: ',
value=42)
widgets.VBox([text_prompt, seed, num_steps])
```

```
{"model_id":"74b4258eac6d4812b53ae3ecf19d43fc","version_major":2,"vers
ion_minor":0}
```

```
import ipywidgets as widgets

text = '\n\t'.join(text_prompt.value.split('.'))
print("Input text:")
print("\t" + text)
display(final_image)
```

```
Input text:
        A brown dog chasing a ball
```

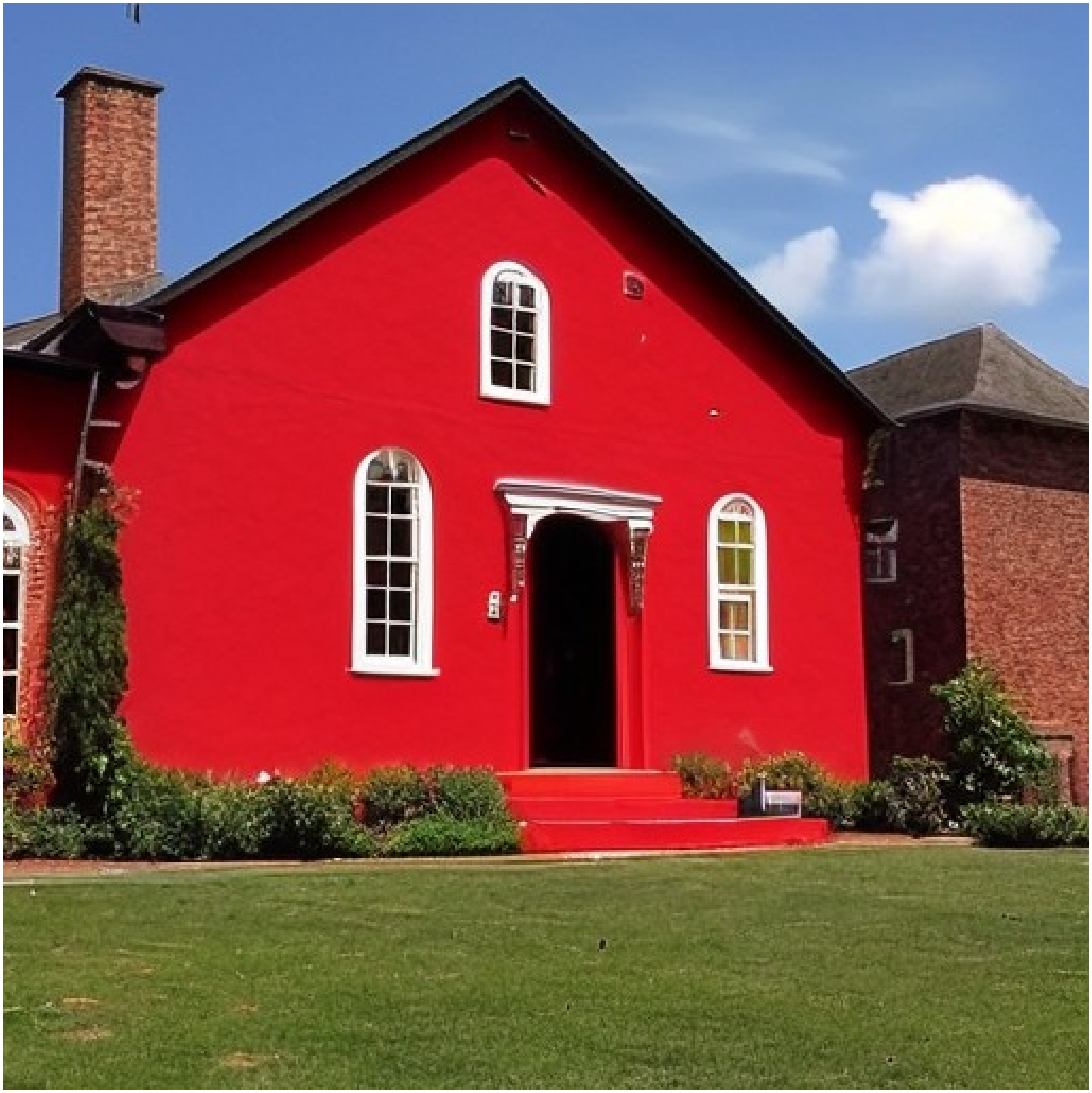
```
import ipywidgets as widgets
sample_text = ('A big house with a red door.' )
text_prompt = widgets.Text(value=sample_text, description='your text')
num_steps = widgets.IntSlider(min=1, max=50, value=20,
description='steps:')
seed = widgets.IntSlider(min=0, max=10000000, description='seed: ',
value=42)
widgets.VBox([text_prompt, seed, num_steps])
```

{"model_id":"711615c45026435191945341d937babc","version_major":2,"version_minor":0}

```python
import ipywidgets as widgets

text = '\n\t'.join(text_prompt.value.split('.'))
print("Input text:")
print("\t" + text)
display(final_image)
```

```
Input text:
	A big house with a red door
	
```



```python
import ipywidgets as widgets
sample_text = ('A yellow flower in a purple pot.' )
text_prompt = widgets.Text(value=sample_text, description='your text')
num_steps = widgets.IntSlider(min=1, max=50, value=20,
description='steps:')
seed = widgets.IntSlider(min=0, max=10000000, description='seed: ',
value=42)
widgets.VBox([text_prompt, seed, num_steps])
```

{"model_id":"37fb74beb3de472083a19a3665ce1971","version_major":2,"version_minor":0}

```python
import ipywidgets as widgets

text = '\n\t'.join(text_prompt.value.split('.'))
print("Input text:")
print("\t" + text)
display(final_image)
```

```
Input text:
	A yellow flower in a purple pot
	
```

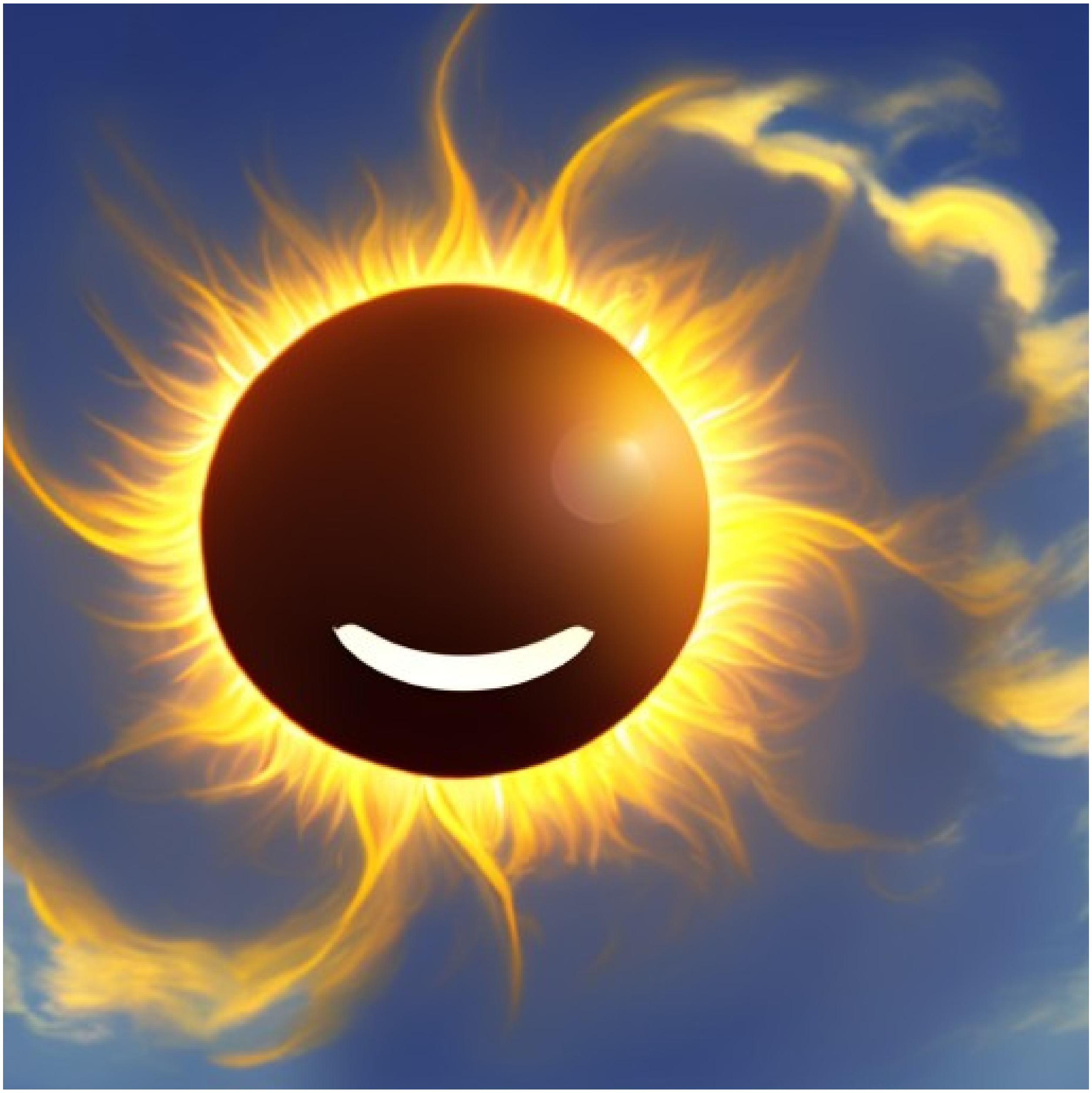
```
import ipywidgets as widgets
sample_text = ('A smiling sun with a face in the sky.' )
text_prompt = widgets.Text(value=sample_text, description='your text')
num_steps = widgets.IntSlider(min=1, max=50, value=20,
description='steps:')
seed = widgets.IntSlider(min=0, max=10000000, description='seed: ',
value=42)
widgets.VBox([text_prompt, seed, num_steps])
```

{"model_id":"1a054fa4bee1403195e6ecbad0779695","version_major":2,"version_minor":0}

```
import ipywidgets as widgets

text = '\n\t'.join(text_prompt.value.split('.'))
print("Input text:")
print("\t" + text)
display(final_image)
```

```
Input text:
	A smiling sun with a face in the sky
	
```



```python
import ipywidgets as widgets
sample_text = ('A happy child playing with a toy.' )
text_prompt = widgets.Text(value=sample_text, description='your text')
num_steps = widgets.IntSlider(min=1, max=50, value=20,
description='steps:')
seed = widgets.IntSlider(min=0, max=10000000, description='seed: ',
value=42)
widgets.VBox([text_prompt, seed, num_steps])
```

{"model_id":"387623fd346e4ac6a27a297aa47a071b","version_major":2,"version_minor":0}

```python
import ipywidgets as widgets

text = '\n\t'.join(text_prompt.value.split('.'))
print("Input text:")
print("\t" + text)
display(final_image)
```

```
Input text:
        A happy child playing with a toy
        
```

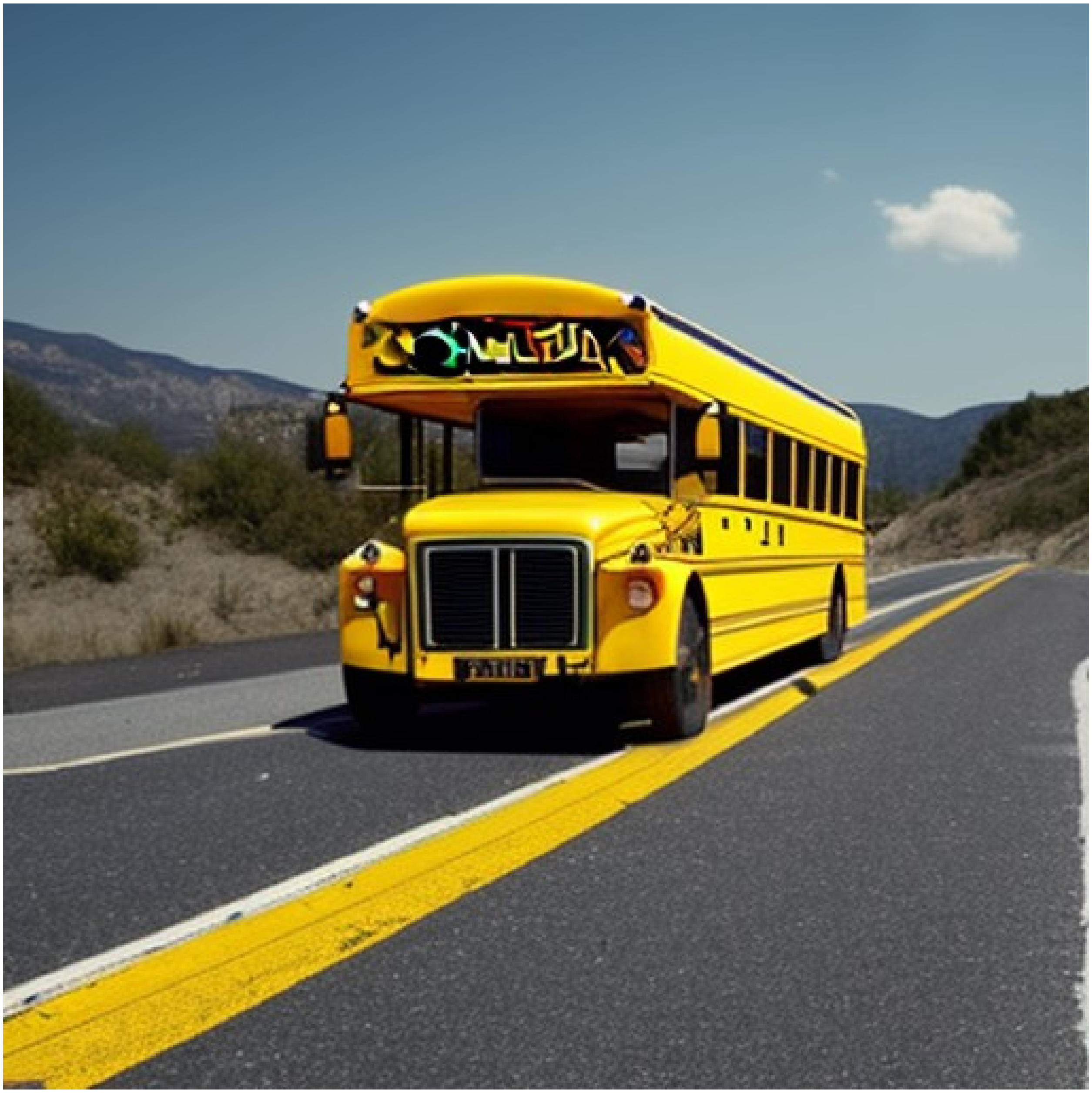
```
import ipywidgets as widgets
sample_text = ('A yellow school bus on a road.' )
text_prompt = widgets.Text(value=sample_text, description='your text')
num_steps = widgets.IntSlider(min=1, max=50, value=20,
description='steps:')
seed = widgets.IntSlider(min=0, max=10000000, description='seed: ',
value=42)
widgets.VBox([text_prompt, seed, num_steps])
```

```
{"model_id":"a33b7bef58f24532873fa5d85f780923","version_major":2,"vers
ion_minor":0}
```

```python
import ipywidgets as widgets

text = '\n\t'.join(text_prompt.value.split('.'))
print("Input text:")
print("\t" + text)
display(final_image)
```

```
Input text:
	A yellow school bus on a road
```



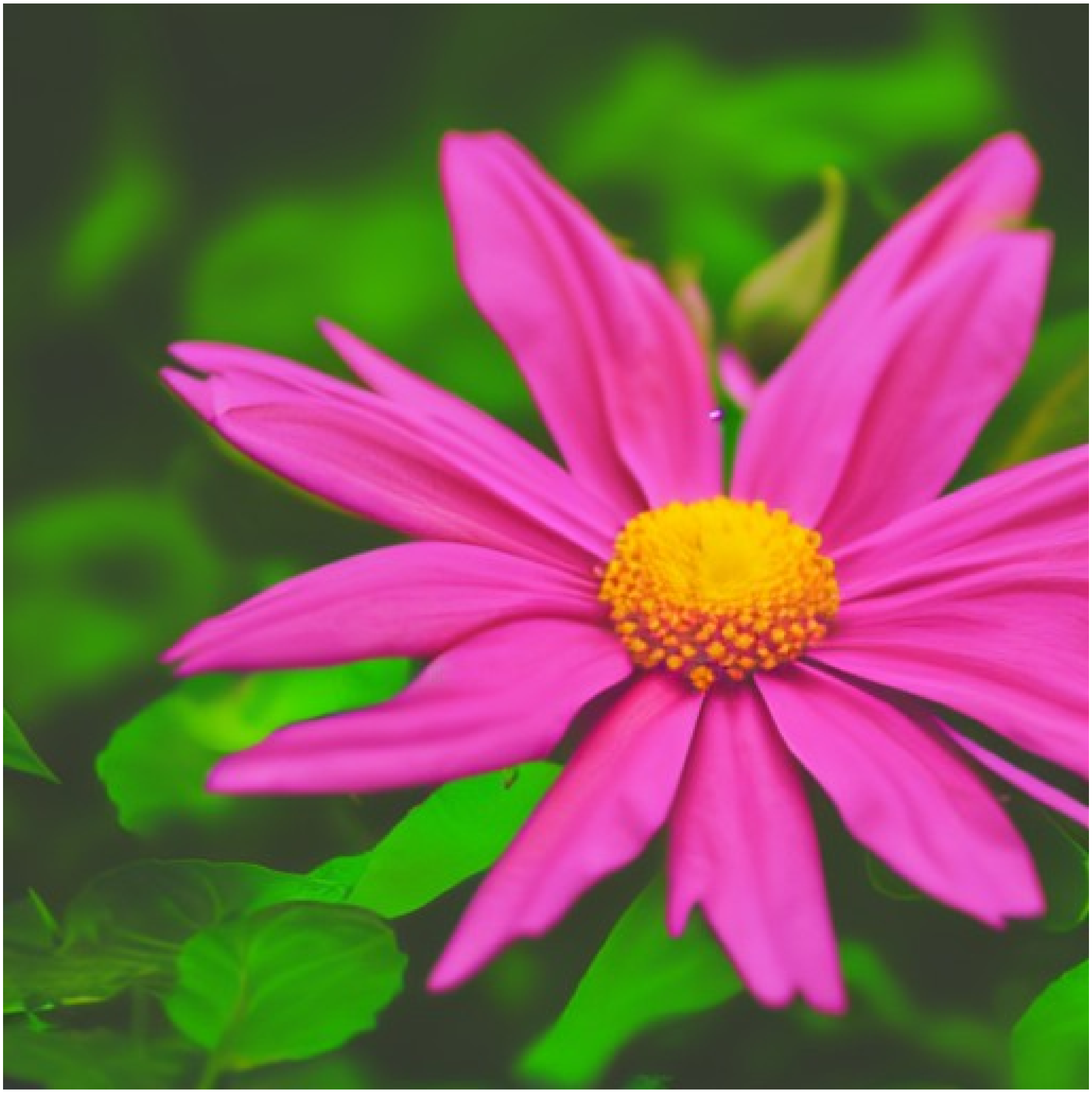
```python
import ipywidgets as widgets
sample_text = ('A pink flower in a green garden.' )
text_prompt = widgets.Text(value=sample_text, description='your text')
num_steps = widgets.IntSlider(min=1, max=50, value=20,
description='steps:')
seed = widgets.IntSlider(min=0, max=10000000, description='seed: ',
value=42)
widgets.VBox([text_prompt, seed, num_steps])
```

{"model_id":"1dccfa3724644b2e81ef70e4bc8f3df2","version_major":2,"version_minor":0}

```python
import ipywidgets as widgets

text = '\n\t'.join(text_prompt.value.split('.'))
print("Input text:")
print("\t" + text)
display(final_image)
```

```
Input text:
	A pink flower in a green garden
	
```



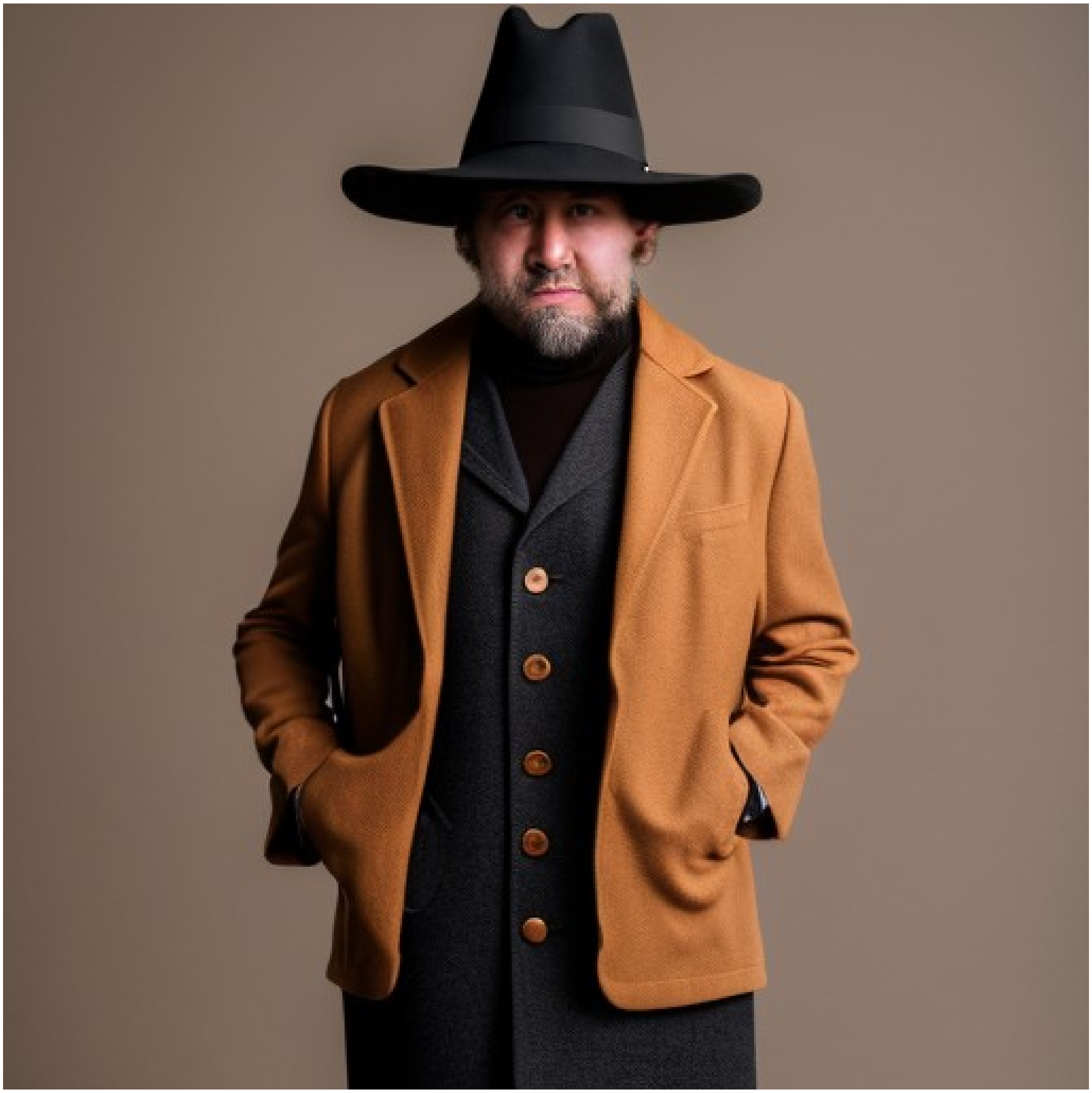
```
import ipywidgets as widgets
sample_text = ('A black hat on a brown coat.' )
text_prompt = widgets.Text(value=sample_text, description='your text')
num_steps = widgets.IntSlider(min=1, max=50, value=20,
description='steps:')
seed = widgets.IntSlider(min=0, max=10000000, description='seed: ',
value=42)
widgets.VBox([text_prompt, seed, num_steps])
```

{"model_id":"cb9ff874b2244271b2d8c80907116843","version_major":2,"version_minor":0}

```python
import ipywidgets as widgets

text = '\n\t'.join(text_prompt.value.split('.'))
print("Input text:")
print("\t" + text)
display(final_image)
```

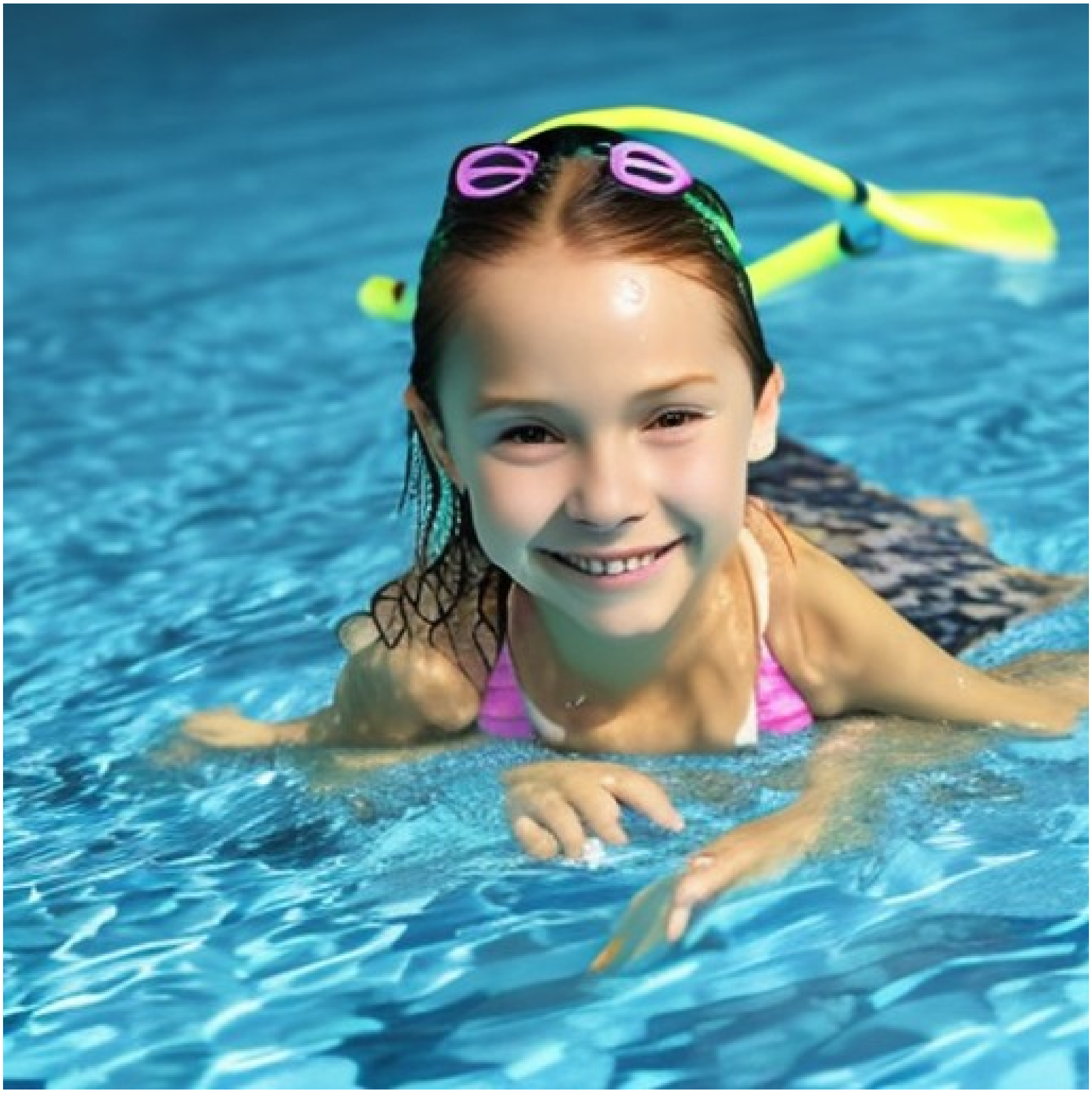
```
Input text:
        A black hat on a brown coat
        
```



```
import ipywidgets as widgets

text = '\n\t'.join(text_prompt.value.split('.'))
print("Input text:")
print("\t" + text)
display(final_image)
```

```
Input text:
	A girl is swimming in a swimming pool
	
```



```python
import ipywidgets as widgets
sample_text = ('Two trains are crossing each other ' )
text_prompt = widgets.Text(value=sample_text, description='your text')
num_steps = widgets.IntSlider(min=1, max=50, value=20,
description='steps:')
seed = widgets.IntSlider(min=0, max=10000000, description='seed: ',
value=42)
widgets.VBox([text_prompt, seed, num_steps])
```

{"model_id":"5861ad6e82e14dbfa22f06bca5c364c5","version_major":2,"version_minor":0}

```python
import ipywidgets as widgets

text = '\n\t'.join(text_prompt.value.split('.'))
print("Input text:")
print("\t" + text)
display(final_image)
```

```
Input text:
	Two trains are crossing each other
```

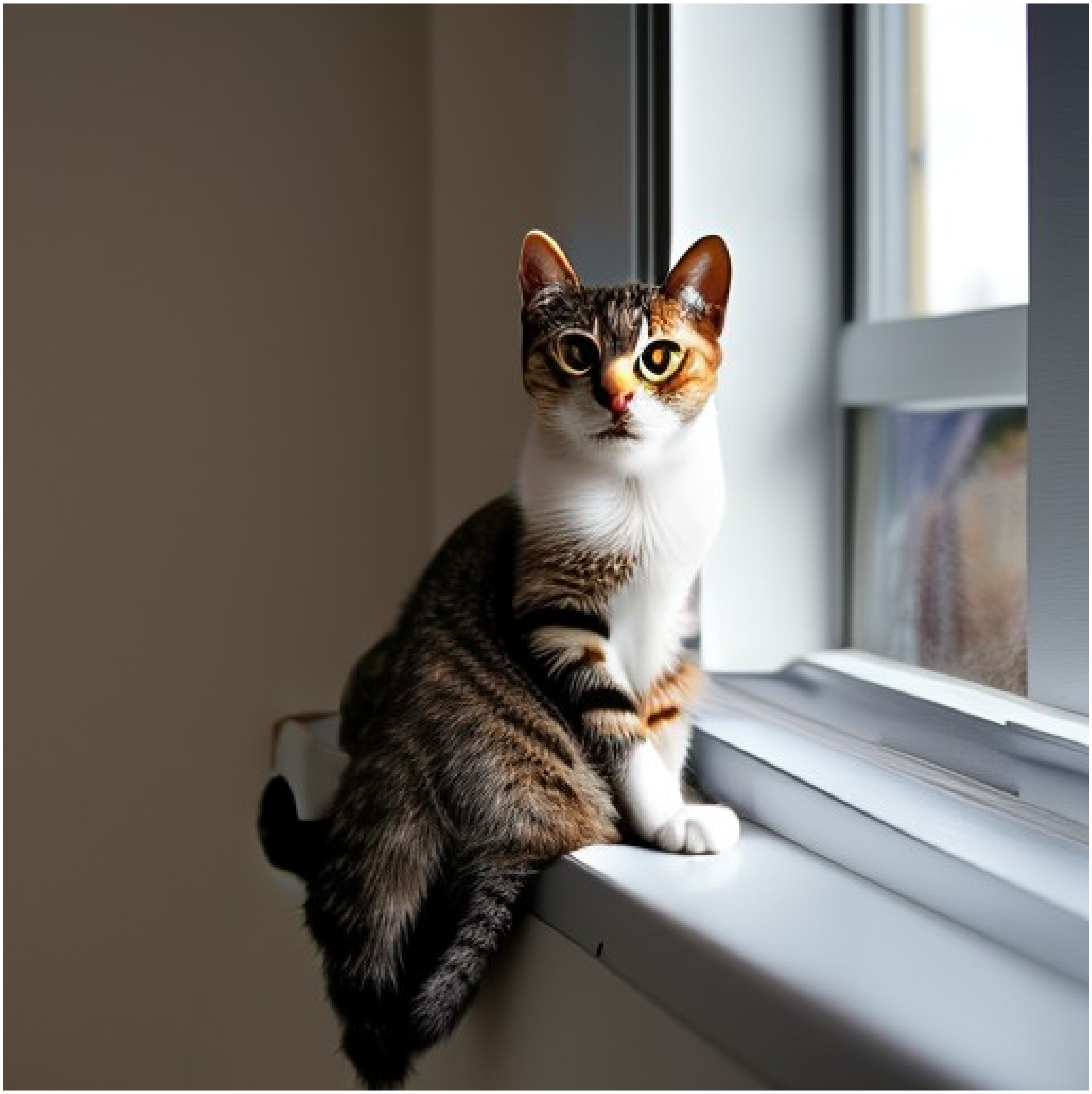
```python
import ipywidgets as widgets
sample_text = ('The cat sat lazily on the windowsill ' )
text_prompt = widgets.Text(value=sample_text, description='your text')
num_steps = widgets.IntSlider(min=1, max=50, value=20,
description='steps:')
seed = widgets.IntSlider(min=0, max=10000000, description='seed: ',
value=42)
widgets.VBox([text_prompt, seed, num_steps])
```

```
{"model_id":"8624207fb0cb4956a135b422dc2aec7b","version_major":2,"version_minor":0}
```

```
import ipywidgets as widgets

text = '\n\t'.join(text_prompt.value.split('.'))
print("Input text:")
print("\t" + text)
display(final_image)
```

```
Input text:
	The cat sat lazily on the windowsill
```



```
import ipywidgets as widgets
sample_text = ('A boy playing with a ball in a swimming pool ' )
```

```
text_prompt = widgets.Text(value=sample_text, description='your text')
num_steps = widgets.IntSlider(min=1, max=50, value=20,
description='steps:')
seed = widgets.IntSlider(min=0, max=10000000, description='seed: ',
value=42)
widgets.VBox([text_prompt, seed, num_steps])
```

{"model_id":"d82d919a21f84b4caa0425c3aaad3f01","version_major":2,"version_minor":0}

```
import ipywidgets as widgets

text = '\n\t'.join(text_prompt.value.split('.'))
print("Input text:")
print("\t" + text)
display(final_image)
```

```
Input text:
	A boy playing with a ball in a swimming pool
```

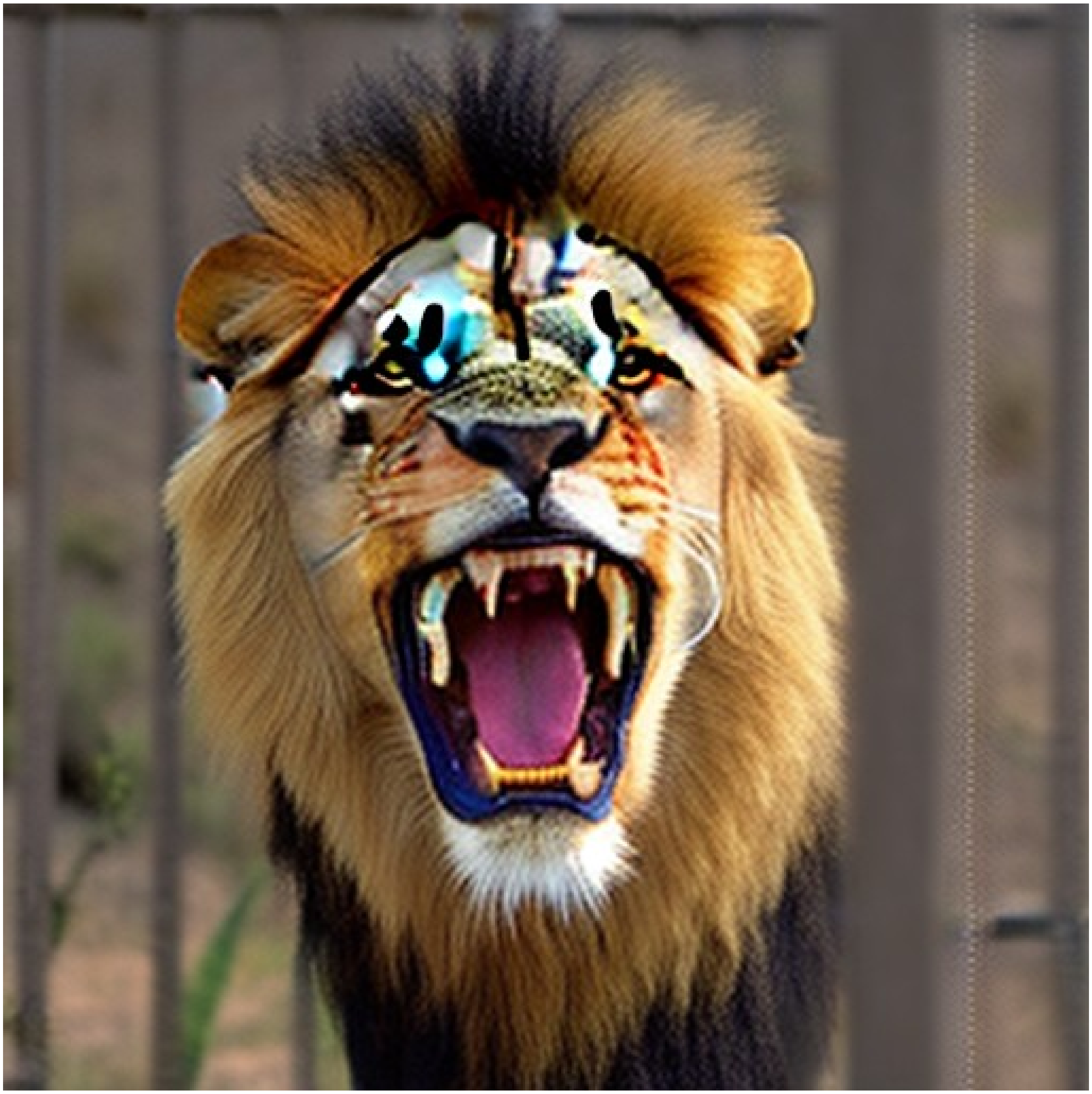
```
import ipywidgets as widgets
sample_text = ('A lion is roaring in the cage ' )
text_prompt = widgets.Text(value=sample_text, description='your text')
num_steps = widgets.IntSlider(min=1, max=50, value=20,
description='steps:')
seed = widgets.IntSlider(min=0, max=10000000, description='seed: ',
value=42)
widgets.VBox([text_prompt, seed, num_steps])
```

{"model_id":"608675cf1e944763b77fc6f47331ea77","version_major":2,"version_minor":0}

```python
import ipywidgets as widgets

text = '\n\t'.join(text_prompt.value.split('.'))
print("Input text:")
print("\t" + text)
display(final_image)
```

```
Input text:
	A lion is roaring in the cage
```



```python
import ipywidgets as widgets
sample_text = ('A group of friends laughed and danced around a bonfire
on a starry summer night. ' )
text_prompt = widgets.Text(value=sample_text, description='your text')
num_steps = widgets.IntSlider(min=1, max=50, value=20,
description='steps:')
seed = widgets.IntSlider(min=0, max=10000000, description='seed: ',
value=42)
widgets.VBox([text_prompt, seed, num_steps])
```

{"model_id":"37f756972d174ceb89fd4d493f85461d","version_major":2,"vers
ion_minor":0}

```
import ipywidgets as widgets

text = '\n\t'.join(text_prompt.value.split('.'))
print("Input text:")
print("\t" + text)
display(final_image)
```

```
Input text:
	A group of friends laughed and danced around a bonfire on a
starry summer night
	
```

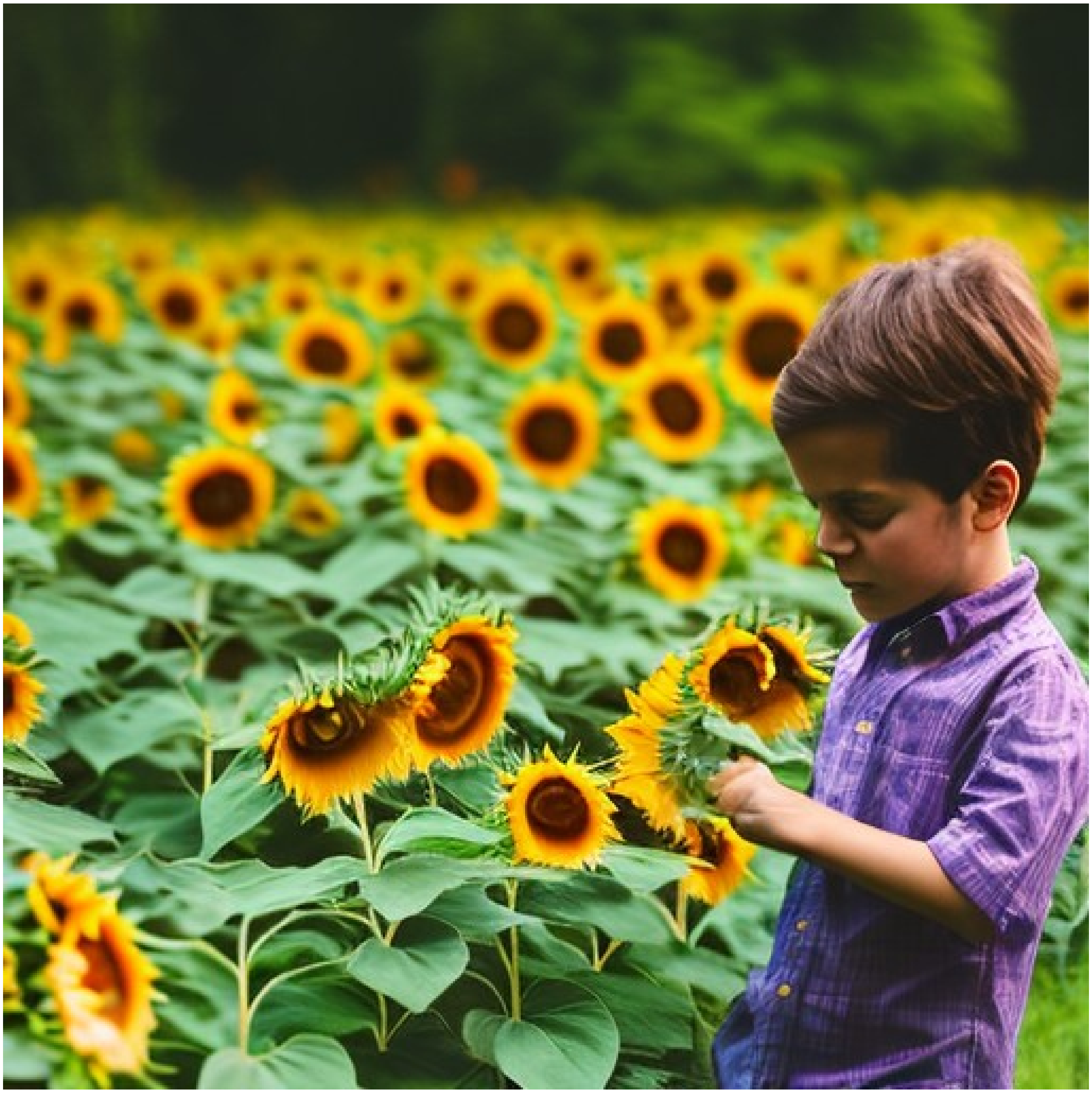
```
import ipywidgets as widgets
sample_text = ('In a lush garden, a young boy with tousled hair and a
curious expression carefully tends to a row of vibrant sunflowers. ' )
text_prompt = widgets.Text(value=sample_text, description='your text')
num_steps = widgets.IntSlider(min=1, max=50, value=20,
description='steps:')
seed = widgets.IntSlider(min=0, max=10000000, description='seed: ',
value=42)
widgets.VBox([text_prompt, seed, num_steps])
```

```
{"model_id":"67bbac615a2142c2922e4a60bbcf2ab7","version_major":2,"vers
ion_minor":0}
```

```python
import ipywidgets as widgets

text = '\n\t'.join(text_prompt.value.split('.'))
print("Input text:")
print("\t" + text)
display(final_image)
```

```
Input text:
	In a lush garden, a young boy with tousled hair and a curious
expression carefully tends to a row of vibrant sunflowers
	
```



```
import ipywidgets as widgets
sample_text = ('A ship in the sea' )
text_prompt = widgets.Text(value=sample_text, description='your text')
num_steps = widgets.IntSlider(min=1, max=50, value=20,
description='steps:')
seed = widgets.IntSlider(min=0, max=10000000, description='seed: ',
value=42)
widgets.VBox([text_prompt, seed, num_steps])
```

{"model_id":"c2578b4f326f42e9bb74e36f47ce51f4","version_major":2,"version_minor":0}

```
import ipywidgets as widgets

text = '\n\t'.join(text_prompt.value.split('.'))
print("Input text:")
print("\t" + text)
display(final_image)
```

```
Input text:
	A ship in the sea
```

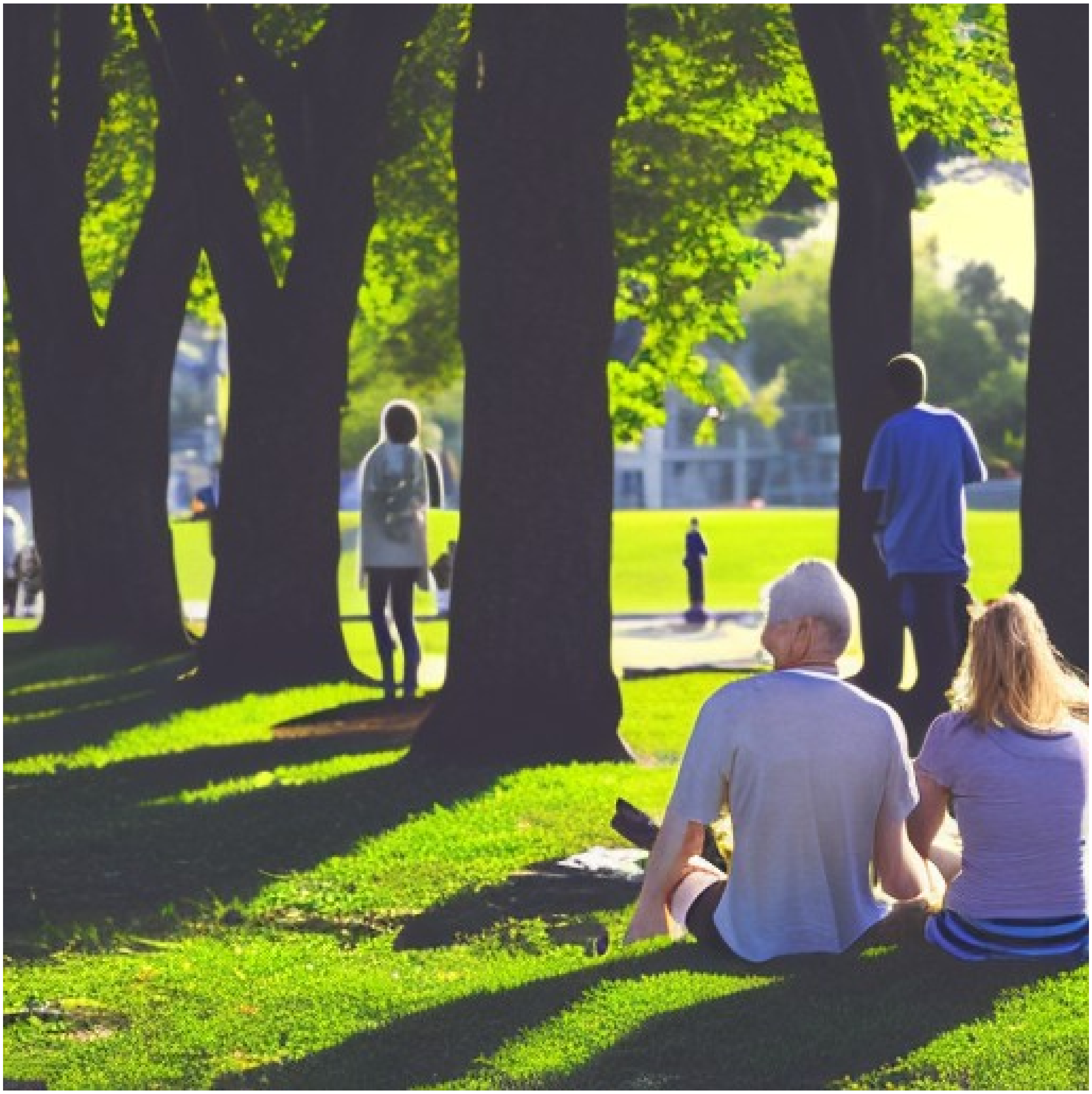
```
import ipywidgets as widgets
sample_text = ('People in a park' )
text_prompt = widgets.Text(value=sample_text, description='your text')
num_steps = widgets.IntSlider(min=1, max=50, value=20,
description='steps:')
seed = widgets.IntSlider(min=0, max=10000000, description='seed: ',
value=42)
widgets.VBox([text_prompt, seed, num_steps])
```

{"model_id":"91119c3635e0493196b8cae48ab3d485","version_major":2,"version_minor":0}

```
import ipywidgets as widgets

text = '\n\t'.join(text_prompt.value.split('.'))
print("Input text:")
print("\t" + text)
display(final_image)

Input text:
        People in a park
```



```
import ipywidgets as widgets
sample_text = ('A penguin dressed like a clown' )
```

```python
text_prompt = widgets.Text(value=sample_text, description='your text')
num_steps = widgets.IntSlider(min=1, max=50, value=20,
description='steps:')
seed = widgets.IntSlider(min=0, max=10000000, description='seed: ',
value=42)
widgets.VBox([text_prompt, seed, num_steps])
```

{"model_id":"16defc7380da41cfb9caf0acc5558809","version_major":2,"version_minor":0}

```python
import ipywidgets as widgets

text = '\n\t'.join(text_prompt.value.split('.'))
print("Input text:")
print("\t" + text)
display(final_image)
```

```
Input text:
	A penguin dressed like a clown
```

```
import ipywidgets as widgets
sample_text = ('Under the canopy of stars, a young girl with a radiant
smile gazes up at the moon, her silhouette illuminated by its soft
glow. ' )
text_prompt = widgets.Text(value=sample_text, description='your text')
num_steps = widgets.IntSlider(min=1, max=50, value=20,
description='steps:')
seed = widgets.IntSlider(min=0, max=10000000, description='seed: ',
value=42)
widgets.VBox([text_prompt, seed, num_steps])
```

{"model_id":"c0658f6b83bc4a85bad9072e5540d2e9","version_major":2,"version_minor":0}

```
import ipywidgets as widgets

text = '\n\t'.join(text_prompt.value.split('.'))
print("Input text:")
print("\t" + text)
display(final_image)
```

```
Input text:
	Under the canopy of stars, a young girl with a radiant smile gazes up at the moon, her silhouette illuminated by its soft glow
	
```

```python
import ipywidgets as widgets
sample_text = ('The teacher enthusiastically explained complex
mathematical concepts to the students, who eagerly absorbed the
knowledge, transforming the classroom into a dynamic hub of learning
and discovery. ' )
text_prompt = widgets.Text(value=sample_text, description='your text')
num_steps = widgets.IntSlider(min=1, max=50, value=20,
description='steps:')
seed = widgets.IntSlider(min=0, max=10000000, description='seed: ',
value=42)
widgets.VBox([text_prompt, seed, num_steps])
```

{"model_id":"fe4e8712c3f44fc38756059cb7651cb6","version_major":2,"vers
ion_minor":0}

```
import ipywidgets as widgets

text = '\n\t'.join(text_prompt.value.split('.'))
print("Input text:")
print("\t" + text)
display(final_image)
```

```
Input text:
     The teacher enthusiastically explained complex mathematical
concepts to the students, who eagerly absorbed the knowledge,
transforming the classroom into a dynamic hub of learning and
discovery
```

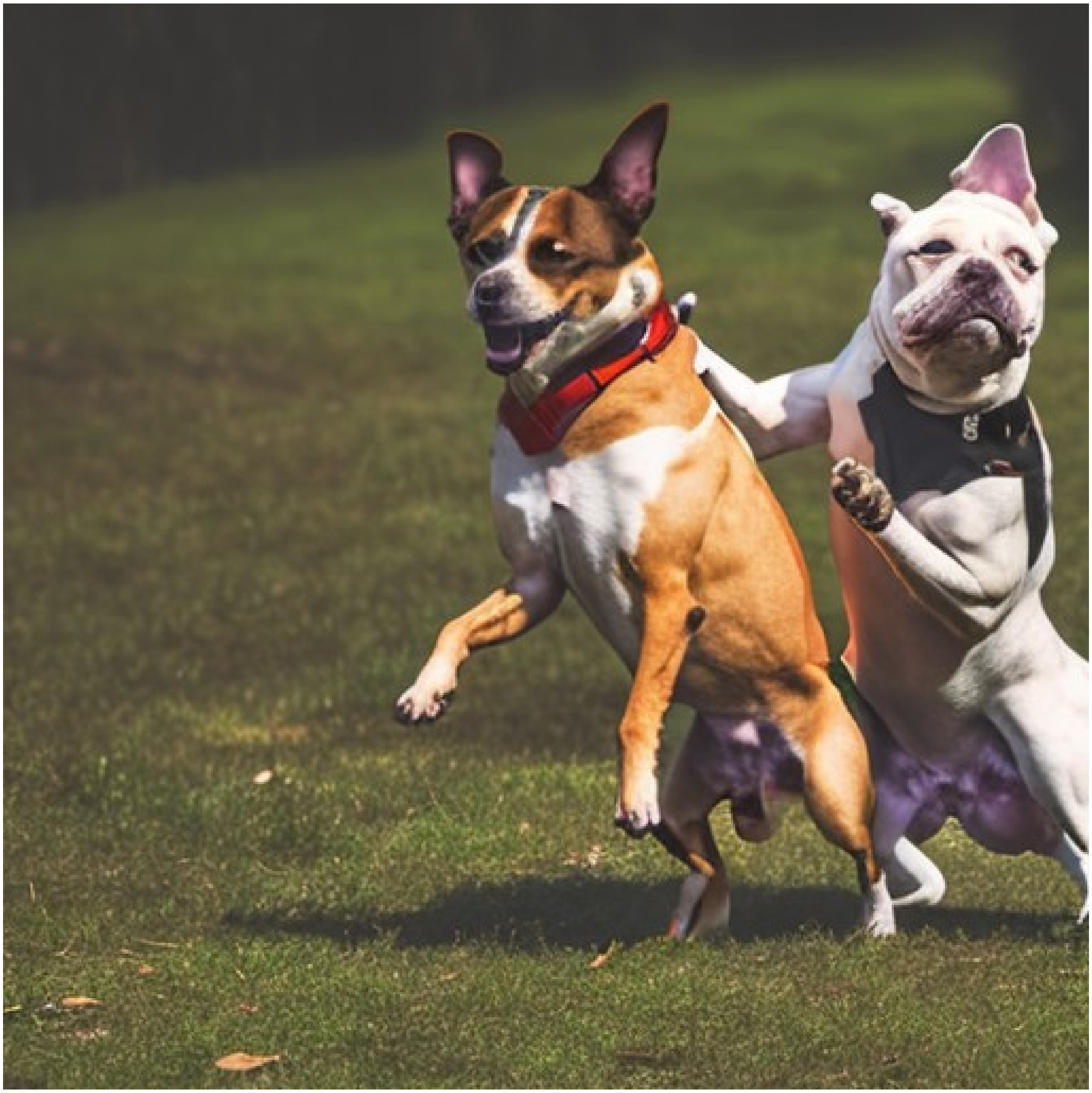
```
import ipywidgets as widgets
sample_text = ('Two dogs are fighting.' )
text_prompt = widgets.Text(value=sample_text, description='your text')
num_steps = widgets.IntSlider(min=1, max=50, value=20,
description='steps:')
seed = widgets.IntSlider(min=0, max=10000000, description='seed: ',
value=42)
widgets.VBox([text_prompt, seed, num_steps])
```

{"model_id":"d0ab1f20bbb240c9926acd83cbdc7d64","version_major":2,"version_minor":0}

```
import ipywidgets as widgets

text = '\n\t'.join(text_prompt.value.split('.'))
print("Input text:")
print("\t" + text)
display(final_image)
```

```
Input text:
	Two dogs are fighting
	
```



```python
import ipywidgets as widgets
sample_text = ('A girl holding umbrella walking on red light street' )
text_prompt = widgets.Text(value=sample_text, description='your text')
num_steps = widgets.IntSlider(min=1, max=50, value=20,
description='steps:')
seed = widgets.IntSlider(min=0, max=10000000, description='seed: ',
value=42)
widgets.VBox([text_prompt, seed, num_steps])
```

{"model_id":"ef2e4c1743ae4db3868bac8b17ac4768","version_major":2,"version_minor":0}

```python
import ipywidgets as widgets

text = '\n\t'.join(text_prompt.value.split('.'))
print("Input text:")
print("\t" + text)
display(final_image)
```

```
Input text:
	A girl holding umbrella walking on red light street
```

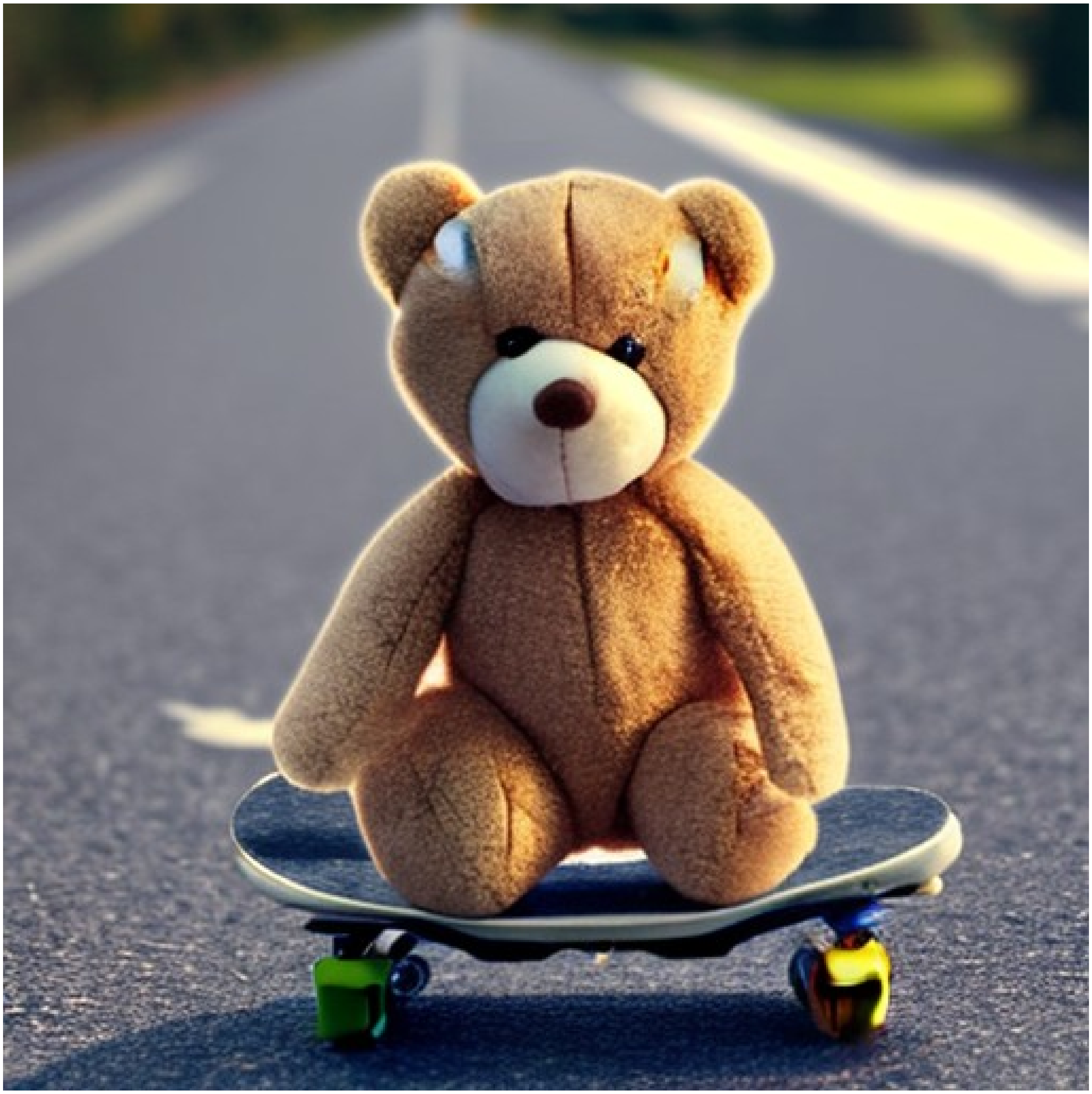
```
import ipywidgets as widgets
sample_text = ('A teddy bear on a skateboard in a road' )
text_prompt = widgets.Text(value=sample_text, description='your text')
num_steps = widgets.IntSlider(min=1, max=50, value=20,
description='steps:')
seed = widgets.IntSlider(min=0, max=10000000, description='seed: ',
value=42)
widgets.VBox([text_prompt, seed, num_steps])
```

```
{"model_id":"e7737a2487d149d282448657858f3a90","version_major":2,"vers
ion_minor":0}
```

```
import ipywidgets as widgets

text = '\n\t'.join(text_prompt.value.split('.'))
print("Input text:")
print("\t" + text)
display(final_image)
```

```
Input text:
     A teddy bear on a skateboard in a road
```



```
import ipywidgets as widgets
sample_text = ('An astronaut walking in a green desert' )
```

```
text_prompt = widgets.Text(value=sample_text, description='your text')
num_steps = widgets.IntSlider(min=1, max=50, value=20,
description='steps:')
seed = widgets.IntSlider(min=0, max=10000000, description='seed: ',
value=42)
widgets.VBox([text_prompt, seed, num_steps])
```

{"model_id":"b87b73376967490495fbc4a55afe833b","version_major":2,"version_minor":0}

```
import ipywidgets as widgets

text = '\n\t'.join(text_prompt.value.split('.'))
print("Input text:")
print("\t" + text)
display(final_image)
```

```
Input text:
	An astronaut walking in a green desert
```