Text-to-Image Generation with Stable Diffusion and OpenVINO™

Stable Diffusion is a text-to-image latent diffusion model created by the researchers and engineers from CompVis, Stability AI and LAION. It is trained on 512x512 images from a subset of the LAION-5B database. This model uses a frozen CLIP ViT-L/14 text encoder to condition the model on text prompts. With its 860M UNet and 123M text encoder. See the model card for more information.

General diffusion models are machine learning systems that are trained to denoise random gaussian noise step by step, to get to a sample of interest, such as an image. Diffusion models have shown to achieve state-of-the-art results for generating image data. But one downside of diffusion models is that the reverse denoising process is slow. In addition, these models consume a lot of memory because they operate in pixel space, which becomes unreasonably expensive when generating high-resolution images. Therefore, it is challenging to train these models and also use them for inference. OpenVINO brings capabilities to run model inference on Intel hardware and opens the door to the fantastic world of diffusion models for everyone!

Model capabilities are not limited text-to-image only, it also is able solve additional tasks, for example text-guided image-to-image generation and inpainting. This tutorial also considers how to run text-guided image-to-image generation using Stable Diffusion.

This notebook demonstrates how to convert and run stable diffusion model using OpenVINO.

Notebook contains the following steps:

1. Convert PyTorch models to ONNX format.
2. Convert ONNX models to OpenVINO IR format, using Model Optimizer tool.
3. Run Stable Diffusion pipeline with OpenVINO.

Prerequisites

The following is needed only if you want to use the original model. If not, you do not have to do anything. Just run the notebook.

> **Note**: The original model (for example, `stable-diffusion-v1-4`) requires you to accept the model license before downloading or using its weights. Visit the stable-diffusion-v1-4 card to read and accept the license before you proceed. To use this diffusion model, you must be a registered user in 🤗 Hugging Face Hub. You will need to use an access token for the code below to run. For more information on access tokens, refer to this section of the documentation. You can login on Hugging Face Hub in notebook environment, using following code:

login to huggingfacehub to get access to pretrained model

from huggingface_hub import notebook_login, whoami

try: whoami() print('Authorization token already provided') except OSError: notebook_login()

```
This tutorial uses a Stable Diffusion model, fine-tuned using images from Midjourney v4 (another popular solution for tex
You can find more details about this model on the [model card](https://huggingface.co/prompthero/openjourney). The same s
```

```
!pip install -r requirements.txt
```

```
ERROR: Could not open requirements file: [Errno 2] No such file or directory: 'requirements.txt'
```

Create Pytorch Models pipeline

StableDiffusionPipeline is an end-to-end inference pipeline that you can use to generate images from text with just a few lines of code.

First, load the pre-trained weights of all components of the model.

```
!pip install diffusers
!pip install transformers
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: diffusers in /usr/local/lib/python3.9/dist-packages (0.14.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.9/dist-packages (from diffusers) (1.22.4)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.9/dist-packages (from diffusers
Requirement already satisfied: requests in /usr/local/lib/python3.9/dist-packages (from diffusers) (2.27.1
Requirement already satisfied: Pillow in /usr/local/lib/python3.9/dist-packages (from diffusers) (8.4.0)
Requirement already satisfied: importlib-metadata in /usr/local/lib/python3.9/dist-packages (from diffuser
Requirement already satisfied: filelock in /usr/local/lib/python3.9/dist-packages (from diffusers) (3.10.7
Requirement already satisfied: huggingface-hub>=0.10.0 in /usr/local/lib/python3.9/dist-packages (from dif
Requirement already satisfied: packaging>=20.9 in /usr/local/lib/python3.9/dist-packages (from huggingface
Requirement already satisfied: tqdm>=4.42.1 in /usr/local/lib/python3.9/dist-packages (from huggingface-hu
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.9/dist-packages (from huggingface-hub
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.9/dist-packages (from
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.9/dist-packages (from importlib-metadat
Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/local/lib/python3.9/dist-packages (from r
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.9/dist-packages (from requests
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.9/dist-packages (from requests->diff
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.9/dist-packages (from reque
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: transformers in /usr/local/lib/python3.9/dist-packages (4.27.4)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.9/dist-packages (from transformer
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.9/dist-packages (from transformers) (4
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.9/dist-packages (from transformers) (
Requirement already satisfied: huggingface-hub<1.0,>=0.11.0 in /usr/local/lib/python3.9/dist-packages (fro
Requirement already satisfied: filelock in /usr/local/lib/python3.9/dist-packages (from transformers) (3.1
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.9/dist-packages (from transformers) (
Requirement already satisfied: requests in /usr/local/lib/python3.9/dist-packages (from transformers) (2.2
Requirement already satisfied: tokenizers!=0.11.3,<0.14,>=0.11.1 in /usr/local/lib/python3.9/dist-packages
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.9/dist-packages (from transform
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.9/dist-packages (from
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.9/dist-packages (from requests->tran
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.9/dist-packages (from reque
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.9/dist-packages (from requests
Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/local/lib/python3.9/dist-packages (from r
```

```
from diffusers import StableDiffusionPipeline

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()

del pipe
```

Fetching 15 files: 100% 15/15 [00:00<00:00, 916.19it/s]

````
Cannot initialize model with low cpu memory usage because `accelerate` was not found in the environment. D
```
pip install accelerate
```
.
/usr/local/lib/python3.9/dist-packages/transformers/models/clip/feature_extraction_clip.py:28: FutureWarni
  warnings.warn(
`text_config_dict` is provided which will be used to initialize `CLIPTextConfig`. The value `text_config["
````

Convert models to OpenVINO Intermediate representation (IR) format

OpenVINO supports PyTorch through export to the ONNX format. You will use `torch.onnx.export` function for obtaining ONNX model. You can learn more in the PyTorch documentation. You need to provide a model object, input data for model tracing and a path for saving the model. Optionally, you can provide the target onnx opset for conversion and other parameters specified in documentation (for example, input and output names or dynamic shapes).

While ONNX models are directly supported by OpenVINO™ runtime, it can be useful to convert them to IR format to take advantage of advanced OpenVINO optimization tools and features. You will use OpenVINO Model Optimizer tool for conversion model to IR format and compression weights to `FP16` format.

The model consists of three important parts:

- Text Encoder for creation condition to generate image from text prompt.
- Unet for step by step denoising latent image representation.
- Autoencoder (VAE) for encdoing input image to latent space (if required) and decoding latent space to image back after generation.

Let us convert each part.

Text Encoder

The text-encoder is responsible for transforming the input prompt, for example, "a photo of an astronaut riding a horse" into an embedding space that can be understood by the U-Net. It is usually a simple transformer-based encoder that maps a sequence of input tokens to a sequence of latent text embeddings.

Input of the text encoder is the tensor `input_ids` which contains indexes of tokens from text processed by tokenizer and padded to maximum length accepted by model. Model outputs are two tensors: `last_hidden_state` - hidden state from the last MultiHeadAttention layer in the model and `pooler_out` - Pooled output for whole model hidden states. You will use `opset_version=14`, because model contains `triu` operation, supported in ONNX only starting from this opset.

```
from pathlib import Path
import torch
!pip install openvino-dev[pytorch,ONNX,tensorflow2]==2022.3.0
```

```
Requirement already satisfied: pandas~=1.3.5 in /usr/local/lib/python3.9/dist-packages (from openvino-de
Requirement already satisfied: openvino==2022.3.0 in /usr/local/lib/python3.9/dist-packages (from openvi
Requirement already satisfied: scipy>=1.8 in /usr/local/lib/python3.9/dist-packages (from openvino-dev[O
Requirement already satisfied: addict>=2.4.0 in /usr/local/lib/python3.9/dist-packages (from openvino-de
Requirement already satisfied: protobuf<4.0.0,>=3.18.1 in /usr/local/lib/python3.9/dist-packages (from o
Requirement already satisfied: fastjsonschema~=2.15.1 in /usr/local/lib/python3.9/dist-packages (from op
Requirement already satisfied: onnx<=1.12,>=1.8.1 in /usr/local/lib/python3.9/dist-packages (from openvi
Requirement already satisfied: tensorflow<=2.9.3,>=2.5 in /usr/local/lib/python3.9/dist-packages (from o
Requirement already satisfied: torchvision<=0.14.0,>=0.9.1 in /usr/local/lib/python3.9/dist-packages (fr
Requirement already satisfied: torch<=1.13.0,>=1.8.1 in /usr/local/lib/python3.9/dist-packages (from ope
Requirement already satisfied: yacs>=0.1.8 in /usr/local/lib/python3.9/dist-packages (from openvino-dev[
Requirement already satisfied: typing-extensions>=3.6.2.1 in /usr/local/lib/python3.9/dist-packages (fro
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.9/dist-packages (from pandas~=1.3.
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.9/dist-packages (from pa
Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/local/lib/python3.9/dist-packages (from
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.9/dist-packages (from requests>=2.
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.9/dist-packages (from reques
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.9/dist-packages (from req
Requirement already satisfied: flatbuffers<2,>=1.12 in /usr/local/lib/python3.9/dist-packages (from tens
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.9/dist-packages (from tensorf
Requirement already satisfied: tensorflow-estimator<2.10.0,>=2.9.0rc0 in /usr/local/lib/python3.9/dist-p
Requirement already satisfied: setuptools in /usr/local/lib/python3.9/dist-packages (from tensorflow<=2.
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.9/dist-packages (from tensorflow<
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.9/dist-packages (from tensorf
Requirement already satisfied: tensorboard<2.10,>=2.9 in /usr/local/lib/python3.9/dist-packages (from te
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.9/dist-packages (from tensorfl
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.9/dist-packages (from tensorflow<=2
Requirement already satisfied: h5py>=2.9.0 in /usr/local/lib/python3.9/dist-packages (from tensorflow<=2
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.9/dist-packages (from tensorfl
Requirement already satisfied: keras-preprocessing>=1.1.1 in /usr/local/lib/python3.9/dist-packages (fro
Requirement already satisfied: gast<=0.4.0,>=0.2.1 in /usr/local/lib/python3.9/dist-packages (from tenso
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.9/dist-packages (from tenso
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.9/dist-pac
Requirement already satisfied: packaging in /usr/local/lib/python3.9/dist-packages (from tensorflow<=2.9
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.9/dist-packages (from tenso
Requirement already satisfied: keras<2.10.0,>=2.9.0rc0 in /usr/local/lib/python3.9/dist-packages (from t
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.9/dist-packages (from tensorflow
```

```
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.9/dist-packages (from google-auth
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/python3.9/dist-packages (from 
Requirement already satisfied: importlib-metadata>=4.4 in /usr/local/lib/python3.9/dist-packages (from m
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.9/dist-packages (from werkzeu
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.9/dist-packages (from importlib-metad
Requirement already satisfied: pyasn1<0.5.0,>=0.4.6 in /usr/local/lib/python3.9/dist-packages (from pyas
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.9/dist-packages (from requests-
```

```python
import gc
from pathlib import Path
import torch

TEXT_ENCODER_ONNX_PATH = Path('text_encoder.onnx')
TEXT_ENCODER_OV_PATH = TEXT_ENCODER_ONNX_PATH.with_suffix('.xml')


def convert_encoder_onnx(xtext_encoder: StableDiffusionPipeline, onnx_path:Path):
    """
    Convert Text Encoder model to ONNX.
    Function accepts pipeline, prepares example inputs for ONNX conversion via torch.export,
    Parameters:
        pipe (StableDiffusionPipeline): Stable Diffusion pipeline
        onnx_path (Path): File for storing onnx model
    Returns:
        None
    """
    if not onnx_path.exists():
        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():
            # infer model, just to make sure that it works
            text_encoder(input_ids)
            # export model to ONNX format
            torch.onnx.export(
                text_encoder,  # model instance
                input_ids,  # inputs for model tracing
                onnx_path,  # output file for saving result
                input_names=['tokens'],  # model input name for onnx representation
                output_names=['last_hidden_state', 'pooler_out'],  # model output names for onnx representa
                opset_version=14  # onnx opset version for export
            )
        print('Text Encoder successfully converted to ONNX')


if not TEXT_ENCODER_OV_PATH.exists():
    convert_encoder_onnx(text_encoder, TEXT_ENCODER_ONNX_PATH)
    !mo --input_model $TEXT_ENCODER_ONNX_PATH --compress_to_fp16
    print('Text Encoder successfully converted to IR')
else:
    print(f"Text encoder will be loaded from {TEXT_ENCODER_OV_PATH}")

del text_encoder
gc.collect()
```

```
Text encoder will be loaded from text_encoder.xml
81
```

U-net

Unet model has three inputs:

- `sample` - latent image sample from previous step. Generation process has not been started yet, so you will use random noise.
- `timestep` - current scheduler step.
- `encoder_hidden_state` - hidden state of text encoder.

Model predicts the `sample` state for the next step.

```
!pip install onnx
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: onnx in /usr/local/lib/python3.9/dist-packages (1.12.0)
Requirement already satisfied: protobuf<=3.20.1,>=3.12.2 in /usr/local/lib/python3.9/dist-packages (from o
Requirement already satisfied: numpy>=1.16.6 in /usr/local/lib/python3.9/dist-packages (from onnx) (1.22.4
Requirement already satisfied: typing-extensions>=3.6.2.1 in /usr/local/lib/python3.9/dist-packages (from
```

```
# !git clone https://github.com/microsoft/vcpkg.git
# %cd vcpkg
# !./bootstrap-vcpkg.bat # For powershell
# !./bootstrap-vcpkg.sh # For bash
# !./vcpkg install onnx
```

```
import numpy as np

UNET_ONNX_PATH = Path('unet/unet.onnx')
UNET_OV_PATH = UNET_ONNX_PATH.parents[1] / 'unet.xml'


def convert_unet_onnx(unet:StableDiffusionPipeline, onnx_path:Path):
    """
    Convert Unet model to ONNX, then IR format.
    Function accepts pipeline, prepares example inputs for ONNX conversion via torch.export,
    Parameters:
        pipe (StableDiffusionPipeline): Stable Diffusion pipeline
        onnx_path (Path): File for storing onnx model
    Returns:
        None
    """
    if not onnx_path.exists():
        # 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))

        # model size > 2Gb, it will be represented as onnx with external data files, you will store it in 
        onnx_path.parent.mkdir(exist_ok=True, parents=True)
        unet.eval()

        with torch.no_grad():
            torch.onnx.export(
                unet,
                (latents, t, encoder_hidden_state), str(onnx_path),
                input_names=['latent_model_input', 't', 'encoder_hidden_states'],
                output_names=['out_sample'],
                opset_version = 11,
                verbose=True
            )
        print('Unet successfully converted to ONNX')


if not UNET_OV_PATH.exists():
    convert_unet_onnx(unet, UNET_ONNX_PATH)
    del unet
    gc.collect()
    !mo --input_model $UNET_ONNX_PATH --compress_to_fp16
    print('Unet successfully converted to IR')
else:
    del unet
    print(f"Unet will be loaded from {UNET_OV_PATH}")
gc.collect()
```

```
/usr/local/lib/python3.9/dist-packages/diffusers/models/unet_2d_condition.py:526: TracerWarning: Convertin
  if any(s % default_overall_up_factor != 0 for s in sample.shape[-2:]):
/usr/local/lib/python3.9/dist-packages/diffusers/models/resnet.py:185: TracerWarning: Converting a tensor
  assert hidden_states.shape[1] == self.channels
/usr/local/lib/python3.9/dist-packages/diffusers/models/resnet.py:190: TracerWarning: Converting a tensor
  assert hidden_states.shape[1] == self.channels
/usr/local/lib/python3.9/dist-packages/diffusers/models/resnet.py:112: TracerWarning: Converting a tensor
  assert hidden_states.shape[1] == self.channels
```

```
/usr/local/lib/python3.9/dist-packages/diffusers/models/resnet.py:125: TracerWarning: Converting a tensor
  if hidden_states.shape[0] >= 64:
/usr/local/lib/python3.9/dist-packages/diffusers/models/unet_2d_condition.py:651: TracerWarning: Convertin
  if not return_dict:
/usr/local/lib/python3.9/dist-packages/torch/onnx/symbolic_helper.py:817: UserWarning: You are trying to e
ONNX's Upsample/Resize operator did not match Pytorch's Interpolation until opset 11. Attributes to determ
We recommend using opset 11 and above for models using this operator.
  warnings.warn(
Unet successfully converted to ONNX
[ INFO ] The model was converted to IR v11, the latest model format that corresponds to the source DL fram
Find more information about API v2.0 and IR v11 at https://docs.openvino.ai/latest/openvino_2_0_transition
[ SUCCESS ] Generated IR version 11 model.
[ SUCCESS ] XML file: /content/unet.xml
[ SUCCESS ] BIN file: /content/unet.bin
Unet successfully converted to IR
0
```

VAE

The VAE model has two parts, an encoder and a decoder. The encoder is used to convert the image into a low dimensional latent representation, which will serve as the input to the U-Net model. The decoder, conversely, transforms the latent representation back into an image.

During latent diffusion training, the encoder is used to get the latent representations (latents) of the images for the forward diffusion process, which applies more and more noise at each step. During inference, the denoised latents generated by the reverse diffusion process are converted back into images using the VAE decoder. When you run inference for text-to-image, there is no initial image as a starting point. You can skip this step and directly generate initial random noise.

As the encoder and the decoder are used independently in different parts of the pipeline, it will be better to convert them to separate models.

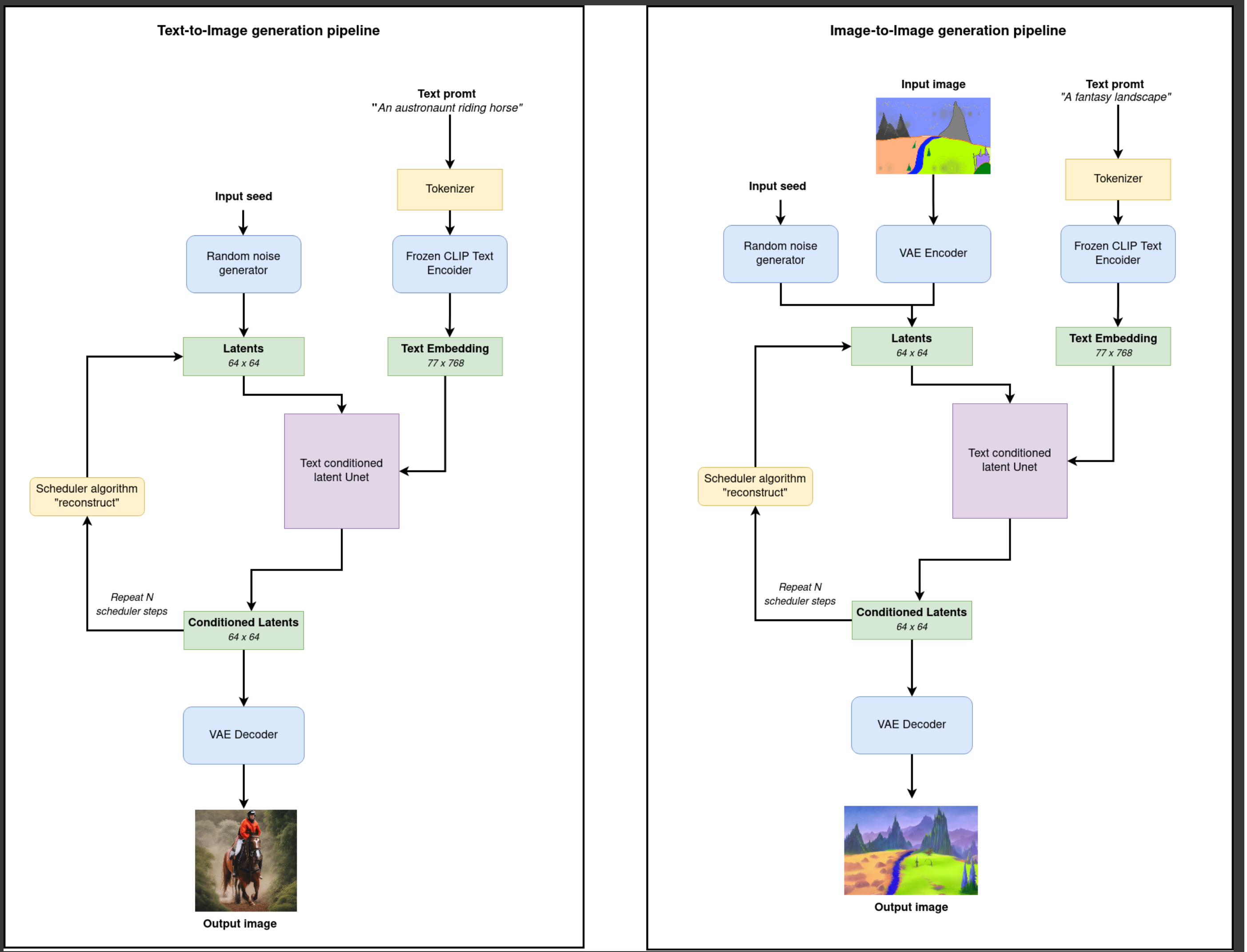
```
VAE_ENCODER_ONNX_PATH = Path('vae_encoder.onnx')
VAE_ENCODER_OV_PATH = VAE_ENCODER_ONNX_PATH.with_suffix('.xml')


def convert_vae_encoder_onnx(vae: StableDiffusionPipeline, onnx_path: Path):
    """
    Convert VAE model to ONNX, then IR format.
    Function accepts pipeline, creates wrapper class for export only necessary for inference part,
    prepares example inputs for ONNX conversion via torch.export,
    Parameters:
        pipe (StableDiffusionInstructPix2PixPipeline): InstrcutPix2Pix pipeline
        onnx_path (Path): File for storing onnx model
    Returns:
        None
    """
    class VAEEncoderWrapper(torch.nn.Module):
        def __init__(self, vae):
            super().__init__()
            self.vae = vae

        def forward(self, image):
            h = self.vae.encoder(image)
            moments = self.vae.quant_conv(h)
            return moments

    if not onnx_path.exists():
        vae_encoder = VAEEncoderWrapper(vae)
        vae_encoder.eval()
        image = torch.zeros((1, 3, 512, 512))
        with torch.no_grad():
            torch.onnx.export(vae_encoder, image, onnx_path, input_names=[
                              'init_image'], output_names=['image_latent'])
        print('VAE encoder successfully converted to ONNX')


if not VAE_ENCODER_OV_PATH.exists():
    convert_vae_encoder_onnx(vae, VAE_ENCODER_ONNX_PATH)
    !mo --input_model $VAE_ENCODER_ONNX_PATH --compress_to_fp16
    print('VAE encoder successfully converted to IR')
else:
```

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

VAE_DECODER_ONNX_PATH = Path('vae_decoder.onnx')
VAE_DECODER_OV_PATH = VAE_DECODER_ONNX_PATH.with_suffix('.xml')


def convert_vae_decoder_onnx(vae: StableDiffusionPipeline, onnx_path: Path):
    """
    Convert VAE model to ONNX, then IR format.
    Function accepts pipeline, creates wrapper class for export only necessary for inference part,
    prepares example inputs for ONNX conversion via torch.export,
    Parameters:
        pipe (StableDiffusionInstructPix2PixPipeline): InstrcutPix2Pix pipeline
        onnx_path (Path): File for storing onnx model
    Returns:
        None
    """
    class VAEDecoderWrapper(torch.nn.Module):
        def __init__(self, vae):
            super().__init__()
            self.vae = vae

        def forward(self, latents):
            latents = 1 / 0.18215 * latents
            return self.vae.decode(latents)

    if not onnx_path.exists():
        vae_decoder = VAEDecoderWrapper(vae)
        latents = torch.zeros((1, 4, 64, 64))

        vae_decoder.eval()
        with torch.no_grad():
            torch.onnx.export(vae_decoder, latents, onnx_path, input_names=[
                              'latents'], output_names=['sample'])
        print('VAE decoder successfully converted to ONNX')


if not VAE_DECODER_OV_PATH.exists():
    convert_vae_decoder_onnx(vae, VAE_DECODER_ONNX_PATH)
    !mo --input_model $VAE_DECODER_ONNX_PATH --compress_to_fp16
    print('VAE decoder successfully converted to IR')
else:
    print(f"VAE decoder will be loaded from {VAE_DECODER_OV_PATH}")

del vae
```

```
/usr/local/lib/python3.9/dist-packages/torch/onnx/_internal/jit_utils.py:258: UserWarning: Constant foldin
  _C._jit_pass_onnx_node_shape_type_inference(node, params_dict, opset_version)
/usr/local/lib/python3.9/dist-packages/torch/onnx/utils.py:687: UserWarning: Constant folding - Only steps
  _C._jit_pass_onnx_graph_shape_type_inference(
/usr/local/lib/python3.9/dist-packages/torch/onnx/utils.py:1178: UserWarning: Constant folding - Only step
  _C._jit_pass_onnx_graph_shape_type_inference(
VAE encoder successfully converted to ONNX
[ INFO ] The model was converted to IR v11, the latest model format that corresponds to the source DL fram
Find more information about API v2.0 and IR v11 at https://docs.openvino.ai/latest/openvino_2_0_transition
[ SUCCESS ] Generated IR version 11 model.
[ SUCCESS ] XML file: /content/vae_encoder.xml
[ SUCCESS ] BIN file: /content/vae_encoder.bin
VAE encoder successfully converted to IR
/usr/local/lib/python3.9/dist-packages/torch/onnx/_internal/jit_utils.py:258: UserWarning: The shape infer
  _C._jit_pass_onnx_node_shape_type_inference(node, params_dict, opset_version)
/usr/local/lib/python3.9/dist-packages/torch/onnx/utils.py:687: UserWarning: The shape inference of prim::
  _C._jit_pass_onnx_graph_shape_type_inference(
/usr/local/lib/python3.9/dist-packages/torch/onnx/utils.py:1178: UserWarning: The shape inference of prim:
  _C._jit_pass_onnx_graph_shape_type_inference(
VAE decoder successfully converted to ONNX
[ INFO ] The model was converted to IR v11, the latest model format that corresponds to the source DL fram
Find more information about API v2.0 and IR v11 at https://docs.openvino.ai/latest/openvino_2_0_transition
[ SUCCESS ] Generated IR version 11 model.
[ SUCCESS ] XML file: /content/vae_decoder.xml
[ SUCCESS ] BIN file: /content/vae_decoder.bin
VAE decoder successfully converted to IR
```

Prepare Inference Pipeline

Putting it all together, let us now take a closer look at how the model works in inference by illustrating the logical flow.



As you can see from the diagram, the only difference between Text-to-Image and text-guided Image-to-Image generation in approach is how initial latent state is generated. In case of Image-to-Image generation, you additionally have an image encoded by VAE encoder mixed with the noise produced by using latent seed, while in Text-to-Image you use only noise as initial latent state. The stable diffusion model takes both a latent image representation of size 64×64 and a text prompt is transformed to text embeddings of size 77×768 via CLIP's text encoder as an input.

Next, the U-Net iteratively *denoises* the random latent image representations while being conditioned on the text embeddings. The output of the U-Net, being the noise residual, is used to compute a denoised latent image representation via a scheduler algorithm. Many different scheduler algorithms can be used for this computation, each having its pros and cons. For Stable Diffusion, it is recommended to use one of:

- PNDM scheduler
- DDIM scheduler
- K-LMS scheduler(you will use it in your pipeline)

Theory on how the scheduler algorithm function works is out of scope for this notebook. Nonetheless, in short, you should remember that you compute the predicted denoised image representation from the previous noise representation and the predicted noise residual. For more information, refer to the recommended Elucidating the Design Space of Diffusion-Based Generative Models

The *denoising* process is repeated given number of times (by default 50) to step-by-step retrieve better latent image representations. When complete, the latent image representation is decoded by the decoder part of the variational auto encoder.

```python
import inspect
from typing import List, Optional, Union, Dict

import PIL
import cv2

from transformers import CLIPTokenizer
from diffusers.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 rati
    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 fit
    then converts it to np.ndarray and adds padding with zeros on right or bottom side of image (depends fr
    converts data to float32 data type and change range of values from [0, 255] to [-1, 1], finally, conver
    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#transformers.CLIPTextModel),
                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_
            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 b
                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 = self.unet.input(0).shape[2] * 8
        self.width = self.unet.input(0).shape[3] * 8
        self.tokenizer = tokenizer

    def __call__(
        self,
        prompt: Union[str, List[str]],
        image: PIL.Image.Image = None,
        num_inference_steps: Optional[int] = 50,
        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
                expense of slower inference.
            guidance_scale (float, *optional*, defaults to 7.5):
                Guidance scale as defined in Classifier-Free Diffusion Guidance(https://arxiv.org/abs/2207.
                guidance_scale is defined as `w` of equation 2.
                Higher guidance scale encourages to generate images that are closely linked to the text pro
                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
                [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.Ima
        """
        if seed is not None:
            np.random.seed(seed)

        if isinstance(prompt, str):
            batch_size = 1
```

```python
        elif isinstance(prompt, list):
            batch_size = len(prompt)
        else:
            raise ValueError(f"`prompt` has to be of type `str` or `list` but is {type(prompt)}")

        img_buffer = []
        # get prompt text embeddings
        text_input = self.tokenizer(
            prompt,
            padding="max_length",
            max_length=self.tokenizer.model_max_length,
            truncation=True,
            return_tensors="np",
        )
        text_embeddings = self.text_encoder(text_input.input_ids)[self._text_encoder_output]
        # here `guidance_scale` is defined analog to the guidance weight `w` of equation (2)
        # of the Imagen paper: https://arxiv.org/pdf/2205.11487.pdf . `guidance_scale = 1`
        # corresponds to doing no classifier free guidance.
        do_classifier_free_guidance = guidance_scale > 1.0
        # get unconditional embeddings for classifier free guidance
        if do_classifier_free_guidance:
            max_length = text_input.input_ids.shape[-1]
            uncond_input = self.tokenizer(
                [""] * batch_size, padding="max_length", max_length=max_length, return_tensors="np"
            )
            uncond_embeddings = self.text_encoder(uncond_input.input_ids)[self._text_encoder_output]

            # For classifier free guidance, you need to do two forward passes.
            # Here you concatenate the unconditional and text embeddings into a single batch
            # to avoid doing two forward passes
            text_embeddings = np.concatenate([uncond_embeddings, text_embeddings])

        # 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), **ext
            if gif:
                image = self.vae_decoder(latents)[self._vae_d_output]
                image = self.postprocess_image(image, meta, output_type)
                img_buffer.extend(image)

```

```python
        # scale and decode the image latents with vae
        image = self.vae_decoder(latents)[self._vae_d_output]

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

    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
        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)
        moments = self.vae_encoder(input_image)[self._vae_e_output]
        mean, logvar = np.split(moments, 2, axis=1)
        std = np.exp(logvar * 0.5)
        latents = (mean + std * np.random.randn(*mean.shape)) * 0.18215
        latents = self.scheduler.add_noise(torch.from_numpy(latents), torch.from_numpy(noise), latent_times
        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
        normalize and convert to [0, 255] pixels range. Optionally, convertes it from np.ndarray to PIL.Ima

        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 imag
               Values that approach 1.0 enable lots of variations but will also produce images that are not
        """
        # 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:]
```

Configure Inference Pipeline

First, you should create instances of OpenVINO Model.

```
from openvino.runtime import Core
core = Core()
text_enc = core.compile_model(TEXT_ENCODER_OV_PATH, 'AUTO')
```

```
unet_model = core.compile_model(UNET_OV_PATH, 'AUTO')
```

```
vae_decoder = core.compile_model(VAE_DECODER_OV_PATH, 'AUTO')
vae_encoder = core.compile_model(VAE_ENCODER_OV_PATH, 'AUTO')
```

Model tokenizer and scheduler are also important parts of the pipeline. Let us define them and put all components together

```
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
)
```

```
Downloading (…)olve/main/vocab.json: 100%  961k/961k [00:00<00:00, 9.61MB/s]
Downloading (…)olve/main/merges.txt: 100%  525k/525k [00:00<00:00, 6.13MB/s]
Downloading (…)cial_tokens_map.json: 100%  389/389 [00:00<00:00, 25.3kB/s]
Downloading (…)okenizer_config.json: 100%  905/905 [00:00<00:00, 62.1kB/s]
```

Text-to-Image generation

Now, you can define a text prompt for image generation and run inference pipeline. Optionally, you can also change the random generator seed for latent state initialization and number of steps.

> **Note**: Consider increasing `steps` to get more precise results. A suggested value is `50`, but it will take longer time to process.

```
import ipywidgets as widgets

text_prompt = widgets.Text(value='A city of Africa with tall buildings. a golden sun shinning', description
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])
```

your text A city of Africa with tall buildings. a

seed: 42

steps: 50

```
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 city of Africa with tall buildings. a golden sun shinning
Seed: 42
Number of steps: 50
```

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

100% 50/50 [02:53<00:00, 3.44s/it]

Finally, let us save generation results. The pipeline returns several results: `sample` contains final generated image, `iterations` contains list of intermediate results for each step.

```
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
final_image.save('result.png')
```

Now is show time!

```
import ipywidgets as widgets

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

```
Input text:
	cyberpunk cityscape like Tokyo New York  with tall buildings at dusk golden hour cinematic lightin
	 A golden daylight, hyper-realistic environment
	 Hyper and intricate detail, photo-realistic
	 Cinematic and volumetric light
	 Epic concept art
	 Octane render and Unreal Engine, trending on artstation
```

```
import ipywidgets as widgets

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

```
Input text:
	A city of Paris with tall buildings
	 a golden sun shinning
```

```
import ipywidgets as widgets

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

```
Input text:
        A city of Africa with tall buildings
         a golden sun shinning
```

0s completed at 02:09