作业 1：

完善 export_onnx.py ,其中已经包含 CLIP 模型的 export model 代码 ,补充 control_net、unet 和 decoder 的 export onnx 代码。

```python
def export_control_net_model():
    """
    导出 ControlNet 模型到 ONNX 格式
    """
    control_net = hk.model.control_model

    # 定义输入数据
    x_noisy = torch.randn(1, 4, 32, 48, dtype=torch.float32)
    hint = torch.randn(1, 3, 256, 384, dtype=torch.float32)
    timestep = torch.tensor([1], dtype=torch.int32)
    context = torch.randn(1, 77, 768, dtype=torch.float32)

    # 定义 ONNX 文件路径
    onnx_path = "./onnx/ControlNet.onnx"

    # 定义输入输出名称
    input_names = ["x_noisy", "hint", "timestep", "context"]
    output_names = ["latent"]

    print("====================== 开始导出 ControlNet ONNX 模型 ======================")

    # 导出 ONNX
    torch.onnx.export(
        control_net,
        (x_noisy, hint, timestep, context),
        onnx_path,
        verbose=True,
        opset_version=18,
        do_constant_folding=True,
        input_names=input_names,
        output_names=output_names,
        keep_initializers_as_inputs=True,
    )

    print("====================== ControlNet ONNX 模型导出完成 ======================")

    # 验证 ONNX
    outputs = control_net(x_noisy, hint, timestep, context)
    input_dicts = {
        "x_noisy": x_noisy.numpy(),
        "hint": hint.numpy(),
        "timestep": timestep.numpy(),
        "context": context.numpy()
    }
    onnxruntime_check(onnx_path, input_dicts, outputs)

    print("====================== ControlNet ONNX 模型验证完成 ======================")
```

```python
def export_controlled_unet_model():
    """
    导出 Unet 模型到 ONNX 格式
    """
    controlled_unet_model = hk.model.model.diffusion_model

    # 定义输入数据
    x_noisy = torch.randn(1, 4, 32, 48, dtype=torch.float32)
    timestep = torch.tensor([1], dtype=torch.int32)
    context = torch.randn(1, 77, 768, dtype=torch.float32)

    # Control 作为一个 list 传入，其中包含 13 个张量
    control_list = [
        torch.randn(1, 320, 32, 48, dtype=torch.float32),
        torch.randn(1, 320, 32, 48, dtype=torch.float32),
        torch.randn(1, 320, 32, 48, dtype=torch.float32),
        torch.randn(1, 320, 16, 24, dtype=torch.float32),
        torch.randn(1, 640, 16, 24, dtype=torch.float32),
        torch.randn(1, 640, 16, 24, dtype=torch.float32),
        torch.randn(1, 640, 8, 12, dtype=torch.float32),
        torch.randn(1, 1280, 8, 12, dtype=torch.float32),
        torch.randn(1, 1280, 8, 12, dtype=torch.float32),
        torch.randn(1, 1280, 4, 6, dtype=torch.float32),
        torch.randn(1, 1280, 4, 6, dtype=torch.float32),
        torch.randn(1, 1280, 4, 6, dtype=torch.float32),
        torch.randn(1, 1280, 4, 6, dtype=torch.float32),
    ]

    # 构造输入和输出的名称
    input_names = ["x_noisy", "timestep", "context"]
    for i in range(0, len(control_list)):
        input_names.append(f"control{i}")

    output_names = ["latent"]

    # 定义 ONNX 存储路径
    onnx_path = "./onnx/ControlledUnet/ControlledUnet.onnx"
    os.makedirs(os.path.dirname(onnx_path), exist_ok=True)

    print("====================== 开始导出 Unet ONNX 模型 ======================")

    # 导出 ONNX
    torch.onnx.export(
        controlled_unet_model,
        (x_noisy, timestep, context, control_list),
        onnx_path,
        verbose=True,
        opset_version=18,
        do_constant_folding=True,
        input_names=input_names,
        # output_names=output_names,
    )

    print("====================== Unet ONNX 模型导出完成 ======================")

    # 验证 ONNX
```

```python
def export_decoder_model():
    """
    导出 Decoder (VAE Decoder) 模型到 ONNX 格式
    """
    decode_model = hk.model.first_stage_model
    decode_model.forward = decode_model.decode  # 直接使用 decode 作为 forward 方法

    # 定义输入
    latent = torch.randn(1, 4, 32, 48, dtype=torch.float32)

    # 定义 ONNX 存储路径
    onnx_path = "./onnx/Decoder.onnx"

    # 定义输入和输出名称
    input_names = ["latent"]
    output_names = ["images"]

    print("====================== 开始导出 Decoder ONNX 模型 ======================")

    # 导出 ONNX
    torch.onnx.export(
        decode_model,
        (latent,),
        onnx_path,
        verbose=True,
        opset_version=18,
        do_constant_folding=True,
        input_names=input_names,
        # output_names=output_names,
        keep_initializers_as_inputs=True,
    )

    print("====================== Decoder ONNX 模型导出完成 ======================")

    # 验证 ONNX
    output = decode_model(latent)
    input_dicts = {"latent": latent.numpy()}
    onnxruntime_check(onnx_path, input_dicts, [output])

    print("====================== Decoder ONNX 模型验证完成 ======================")
```

导出 onnx 模型：

```
root@nv4090-server101:/home/player/TensorRT-StableDiffusion-export_onnx/onnx# ls
CLIP.onnx  ControlNet.onnx  ControlledUnet  Decoder.onnx  readme.md
```

作业 2：

将 export_onnx.py 和 onnx2trt.py 代码合并，化繁为简，并修改所有 dynamic shape 的逻辑，使其变为 static shape。

```python
def export_controlled_unet_model():
    """
    静态 shape:
      - x_noisy: (1, 4, 32, 48)
      - timestep: (1,)
      - context: (1, 77, 768)
      - control_list (13 个 tensor)
    """
    onnx_path = "./onnx/ControlledUnet/ControlledUnet.onnx"
    plan_path = "./engine/ControlledUnet.plan"

    static_shapes = [
        (1, 4, 32, 48),  # x_noisy
        (1,),  # timestep
        (1, 77, 768),  # context
        (1, 320, 32, 48),  # control_0
        (1, 320, 32, 48),  # control_1
        (1, 320, 32, 48),  # control_2
        (1, 320, 16, 24),  # control_3
        (1, 640, 16, 24),  # control_4
        (1, 640, 16, 24),  # control_5
        (1, 640, 8, 12),  # control_6
        (1, 1280, 8, 12),  # control_7
        (1, 1280, 8, 12),  # control_8
        (1, 1280, 4, 6),  # control_9
        (1, 1280, 4, 6),  # control_10
        (1, 1280, 4, 6),  # control_11
        (1, 1280, 4, 6),  # control_12
    ]

    onnx2trt(onnx_path, plan_path, static_shapes)
```

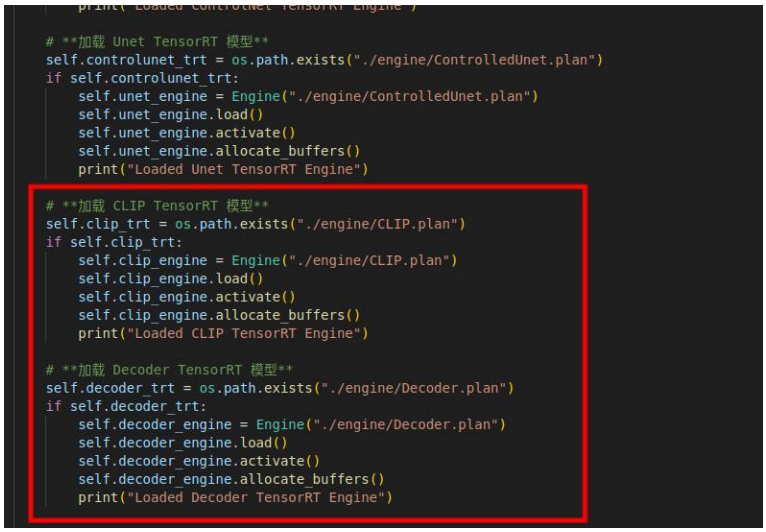
导出静态 shape 的 trt plan 文件：

```
root@nv4090-server101:/home/player/TensorRT-StableDiffusion-export_onnx/engine# ls
CLIP.plan  ControlNet.plan  ControlledUnet.plan  Decoder.plan  readme.md
```

作业 3：

在 cldm_trt 目录下，已经包含了 将 control_net、unet 从 Torch 模型转换为 TensorRT 模型的代码，并补充 CLIP 和 decoder 模型的替换代码。

（调用 trt model 的部分代码不太会写）



```
print("Loaded ControlNet TensorRT Engine")

# **加载 Unet TensorRT 模型**
self.controlunet_trt = os.path.exists("./engine/ControlledUnet.plan")
if self.controlunet_trt:
    self.unet_engine = Engine("./engine/ControlledUnet.plan")
    self.unet_engine.load()
    self.unet_engine.activate()
    self.unet_engine.allocate_buffers()
    print("Loaded Unet TensorRT Engine")

# **加载 CLIP TensorRT 模型**
self.clip_trt = os.path.exists("./engine/CLIP.plan")
if self.clip_trt:
    self.clip_engine = Engine("./engine/CLIP.plan")
    self.clip_engine.load()
    self.clip_engine.activate()
    self.clip_engine.allocate_buffers()
    print("Loaded CLIP TensorRT Engine")

# **加载 Decoder TensorRT 模型**
self.decoder_trt = os.path.exists("./engine/Decoder.plan")
if self.decoder_trt:
    self.decoder_engine = Engine("./engine/Decoder.plan")
    self.decoder_engine.load()
    self.decoder_engine.activate()
    self.decoder_engine.allocate_buffers()
    print("Loaded Decoder TensorRT Engine")
```