SD torch转onnx

分析

入口：compute_score

for函数内产生新图片的函数是

```
hk.process()
```

往上找：

```
from canny2image_TRT import hackathon
hk = hackathon()
```

定位到canny2image

canny2img.py

上网查阅资料，这个函数应该是直接用于：

1.canny边缘检测

2.生成条件输入

3.控制生成结果

原图	条件输入	输出
无		
无		
		

根据compute_score.py中的调用：

```python
new_img = hk.process(img,
            "a bird",
            "best quality, extremely detailed",
            "longbody, lowres, bad anatomy, bad hands, missing fingers",
            1,
            256,
            20,
            False,
            1,
            9,
            2946901,
            0.0,
            100,
            200)
```

分析其核心函数process输入参数如下：

- input_image：输入图片数据，重要
- prompt：描述图像生成的文本提示
- a_prompt："addtional prompt"或者是"activation prompt"，是与prompt配合的附加文本提示，用于调整细节
- n_prompt："negative promot"，负面提示，描述模型在生成中需要规避的内容
- num_samples：生成样本数量
- image_resolution：生成图片的解析度
- ddim_steps：DDIM (Denoising Diffusion Implicit Models) 算法生成图像的迭代次数
- guess_mode：是否开启猜测模式
- strength：控制输入图像的影响力。若生成是基于已有图像，`strength` 决定了参考图像与生成结果的结合程度。`strength=0` 时，完全不参考输入图像，`strength=1` 时，完全参考输入图像。
- scale：控制文本提示对生成图像的影响程度。在大多数图像生成模型中，`scale` 是 `prompt` 的影响力调节因子。较大的 `scale` 会让图像更强烈地遵循提示内容，较小的 `scale` 会使生成过程更自由。
- seed：是否需要设置随机数
- eta：通常与采样策略中的噪声调节有关。DDIM采样方法中，`eta` 调节生成过程中噪声的大小，影响生成图像的细节与噪声水平。
- low_threshold 和 high_threshold：限制图像的亮度、对比度或者像素强度范围，控制生成图像的细节和质量。与图像生成过程中的图像过滤、边缘控制或去噪过程有关。

接下来对process方法进行主句分析

输入图片处理

```python
img = resize_image(HWC3(input_image), image_resolution)
H, W, C = img.shape

detected_map = self.apply_canny(img, low_threshold, high_threshold)
detected_map = HWC3(detected_map)

control = torch.from_numpy(detected_map.copy()).float().cuda() / 255.0
control = torch.stack([control for _ in range(num_samples)], dim=0)
control = einops.rearrange(control, 'b h w c -> b c h w').clone()
```

- 先看HWC3，路径/annotator/util.py

```python
def HWC3(x):
```

```
    assert x.dtype == np.uint8
    if x.ndim == 2:
        x = x[:, :, None]
    assert x.ndim == 3
    H, W, C = x.shape
    assert C == 1 or C == 3 or C == 4
    if C == 3:
        return x
    if C == 1:
        return np.concatenate([x, x, x], axis=2)
    if C == 4:
        color = x[:, :, 0:3].astype(np.float32)
        alpha = x[:, :, 3:4].astype(np.float32) / 255.0
        y = color * alpha + 255.0 * (1.0 - alpha)
        y = y.clip(0, 255).astype(np.uint8)
        return y
```

它应该是一个生成RGB图像的方法。

当为灰度图像（通道数1），将单通道扩展为3通道。

如果是RGBA图像（通道数4），带有透明通道，则使用透明通道融合RGB颜色和背景。简单来说就是透明通道就是设置透明度，当透明度越高RGB 颜色越明显，透明度越低，背景越明显（也就是白色255）。

- 再看resize_image()方法，也位于/annotator/util.py中

```
def resize_image(input_image, resolution):
    H, W, C = input_image.shape
    H = float(H)
    W = float(W)
    k = float(resolution) / min(H, W)
    H *= k
    W *= k
    H = int(np.round(H / 64.0)) * 64
    W = int(np.round(W / 64.0)) * 64
    img = cv2.resize(input_image, (W, H), interpolation=cv2.INTER_LANCZOS4 if k > 1 else cv2.INTER_AREA)
    return img
```

k是输出图片与输入图片最小边比值，保证整形后的图片最小边至少满足输出图片解析度。即根据目标尺寸调整输入图片大小

之后乘除64是为了让图片变为64的倍数，因为网络的输入是64*64。

用了cv2.resize函数进行放缩，其中：

(W, H)是调整后的宽高

后面是使用的方法，如果k>1说明要放大，则可用差值法cv2.INTER_LANCZOS4；如果k<1则说明要缩小，使用cv2.INTER_AREA。

- apply_canny = CannyDetector()，来自from annotator.canny import CannyDetector

```
class CannyDetector:
    def __call__(self, img, low_threshold, high_threshold):
        return cv2.Canny(img, low_threshold, high_threshold)
```

本质是调用了OpenCV的边缘检测算法，获得边缘提取图像

- 模型输入数据处理

control是将处理后的图像放在GPU上加速获得的副本，根据生成图像的数量转为多份拷贝，放在一个张量里。以第0维度堆叠，形成(num_samples, H, W, C)张量。num_samples可以理解为批大小batch, `h` 是图像的高度，`w` 是图像的宽度，`c` 是图像的通道数。

模型创建（初始化中）

```
self.apply_canny = CannyDetector()
        self.model = create_model('./models/cldm_v15.yaml').cpu()
        self.model.cond_stage_model.cuda()
        self.use_trt = True
        # if not self.use_trt:
        if 1:
 self.model.load_state_dict(load_state_dict('./models/control_sd15_canny.pth', location='cuda'))
            self.model = self.model.cuda()


        self.ddim_sampler = DDIMSampler(self.model)
        self.warm_up()
```

- create_model，来自from cldm_trt.model import create_model

```
def create_model(config_path):
    config = OmegaConf.load(config_path)
    model = instantiate_from_config(config.model).cpu()
    print(f'Loaded model config from [{config_path}]')
    return model
```

使用OmegaConf库加载配置文件（.yaml格式），之后根据文件进行模型实例化。

```
def instantiate_from_config(config):
    if not "target" in config:
        if config == '__is_first_stage__':
            return None
        elif config == "__is_unconditional__":
            return None
        raise KeyError("Expected key `target` to instantiate.")
    return get_obj_from_str(config["target"])(**config.get("params", dict()))
```

`"target"` 键通常包含需要实例化的对象的类名或对象的路径。如果配置对象没有"target"键，说明无法知道要实例化的对象是什么，因此会进入后续的错误处理逻辑。有两中情况会跳过模型实例化：

当 `config == '__is_first_stage__'`，用于表示模型第一阶段

当 `config == "__is_unconditional__"`，用于表示某种无条件状态

当有"target"键会进行实例化：

- `config["target"]`：获取 `"target"` 键值，这里是cldm.cldm.ControlLDM
- `get_obj_from_str(config["target"])`：返回cldm.cldm.ControlLDM对应的类或对象

- `**config.get("params", dict())`：以字典形式获取配置中参数，如果没有"params"，则返回一个空字典。**表示将结果作为关键字参数传递给目标类的构造函数。

启用内存优化模式

```
if config.save_memory:
    self.model.low_vram_shift(is_diffusing=False)
```

找到low_vram_shift()函数:

```
def low_vram_shift(self, is_diffusing):
    if is_diffusing:
        self.model = self.model.cuda()
        self.control_model = self.control_model.cuda()
        self.first_stage_model = self.first_stage_model.cpu()
        self.cond_stage_model = self.cond_stage_model.cpu()
    else:
        self.model = self.model.cpu()
        self.control_model = self.control_model.cpu()
        self.first_stage_model = self.first_stage_model.cuda()
        self.cond_stage_model = self.cond_stage_model.cuda()
```

GPU资源有限，并且在不同阶段，O型不同部分可能有不同的计算需求。

因此在"diffusing"阶段

- model和control_model是计算密集型部分。扩散过程包括生成、噪声添加、逐步更新等操作。这些操作通常需要大量的计算资源，因此将它们放在 GPU 上能够利用 GPU 的并行计算能力，从而加速计算过程。
- `first_stage_model` 和 `cond_stage_model` 在扩散阶段，这些模型可能只需要偶尔使用，因此将它们移到 CPU 上可以释放 GPU 显存，避免不必要的显存占用。

在非"diffusing"阶段

- `first_stage_model` **和** `cond_stage_model` 这两个模型在其他阶段可能需要更多的计算。例如，在模型的训练、推理、条件生成等过程中，`first_stage_model` 和 `cond_stage_model` 可能参与了更多的计算任务。这时候它们需要频繁与其他模型交互，因此被放到 GPU 上可以加速这些计算过程。
- `model` **和** `control_model` 在非扩散阶段可能不是计算的核心部分，因此可以将它们移回 CPU 上，节省 GPU 显存，以便其他计算密集型任务使用。

设置随机数种子

```
if seed == -1:
    seed = random.randint(0, 65535)
    seed_everything(seed)
```

拼接条件语句

```
cond = {"c_concat": [control], "c_crossattn": [self.model.get_learned_conditioning([prompt + ', ' + a_prompt] * num_samples)]}
un_cond = {"c_concat": None if guess_mode else [control], "c_crossattn": [self.model.get_learned_conditioning([n_prompt] * num_samples)]}
shape = (4, H // 8, W // 8)
```

控制比重

```
self.model.control_scales = [strength * (0.825 ** float(12 - i)) for i in range(13)] if guess_mode else ([strength] * 13)
```

得到输入图片对生成图片的影响度

如果guss_mode打开，则按照[strength * (0.825 ** float(12 - i)) for i in range(13)]公式生成

如果guss_mode关闭，表示完全由参考输入图片，则control_scales全为1.

参数进入模型

调试：[python pdb 代码调试 - 最全最详细的使用说明 - 简书]

ControlNet

```
samples, intermediates = self.ddim_sampler.sample_simple(ddim_steps, num_samples,
                                                         shape, cond, verbose=False, eta=eta,
                                                         unconditional_guidance_scale=scale,
                                                         unconditional_conditioning=un_cond)
x_samples = self.model.decode_first_stage(samples)
x_samples = (einops.rearrange(x_samples, 'b c h w -> b h w c') * 127.5 + 127.5).cpu().numpy().clip(0, 255).astype(np.uint8)
```

可见调用了self.ddim_sampler的sample_simple()函数

self.ddim_sampler来自self.ddim_sampler = DDIMSampler(self.model)，找到ddim_hacker.py，进行调试。

进入循环：

```
for i, step in enumerate(iterator):
    index = total_steps - i - 1
    ts = torch.full((batch_size,), step, device=device, dtype=torch.long)
```

之后，出现了

```
(Pdb) n
DDIM Sampler:   0%|                                                    | 0/20 [00:00<?, ?it/s]
```

说明开始进入模型输入阶段了。

看到代码

```
if self.controlnet_trt and self.controlunet_trt:
```

```
    hint = torch.cat(conditioning['c_concat'], 1)
    cond_txt = torch.cat(conditioning['c_crossattn'], 1)
    #if self.cuda_graph_instance is None:
    #cudart.cudaStreamBeginCapture(self.stream1.ptr, cudart.cudaStreamCaptureMode.cudaStreamCaptureModeGlobal)

    torch.cuda.synchronize()
    start_time = time.time()
    control_trt_dict = self.controlnet_engine.infer({"x_noisy":img, "hint":hint, "timestep":ts, "context":cond_txt}, stream = self.stream1, use_cuda_graph=True)
    torch.cuda.synchronize()
    end_time = time.time()
    # print(f"controlnet_engine time={(end_time-start_time)*1000}ms")
    control = list(control_trt_dict.values())
    input_dict = {'x_noisy': img, 'timestep': ts, 'context': cond_txt,
                  'control0': control[4], 'control1': control[5], 'control2': control[6], 'control3': control[7],
                  'control4': control[8], 'control5': control[9], 'control6': control[10], 'control7': control[11],
                  'control8': control[12], 'control9': control[13], 'control10': control[14], 'control11': control[15],
                  'control12': control[16]}
    torch.cuda.synchronize()
    start_time = time.time()
    model_t = self.unet_engine.infer(input_dict, self.stream1, use_cuda_graph=True)['latent'].clone()
    torch.cuda.synchronize()
    end_time = time.time()
    # print(f"unet_engine time={(end_time-start_time)*1000}ms")
    else:
        model_t = self.model.apply_model(img, ts, conditioning)
```

看到了'x_noisy'、'timestep'等输入，与老师提示的一致，本以为找到了controlunet的接口，结果pdb进入了else：

> 这里不知道if self.controlnet_trt and self.controlunet_trt判断依据是什么，我的猜测是会否使用TRT的两种模型
>
> 答：从代码
>
> ```
> controlnet_engine_path = "./engine/ControlNet.plan"
> if not os.path.exists(controlnet_engine_path):
>  self.controlnet_trt = False
> ```
>
> 可以看出，如果ControlNet的TRT文件不存在时，就设为False。
>
> 也就是说，在无模型情况下运行会去创建模型，如果在有TRT模型情况下就会运行指定的模型

```
    else:
        model_t = self.model.apply_model(img, ts, conditioning)
```

这里链接到了cldm.py的apply_model，也与老师提示的一致：

```
def apply_model(self, x_noisy, t, cond, *args, **kwargs):
    assert isinstance(cond, dict)
```

```python
    diffusion_model = self.model.diffusion_model

    cond_txt = torch.cat(cond['c_crossattn'], 1)

    if cond['c_concat'] is None:
        eps = diffusion_model(x=x_noisy, timesteps=t, context=cond_txt, control=None, only_mid_control=self.only_mid_control)
        else:
            control = self.control_model(x=x_noisy, hint=torch.cat(cond['c_concat'], 1), timesteps=t, context=cond_txt)
            control = [c * scale for c, scale in zip(control, self.control_scales)]
            eps = diffusion_model(x=x_noisy, timesteps=t, context=cond_txt, control=control, only_mid_control=self.only_mid_control)

            return eps
```

这里最主要的是if判断，从之前的输入：

```python
# canny2image_TRT.py
cond = {"c_concat": [control], "c_crossattn": [self.model.get_learned_conditioning([prompt + ', ' + a_prompt] * num_samples)]}
```

并且用pdb验证了一下：

```
(Pdb) p cond_txt
tensor([[[-0.3884,  0.0229, -0.0522,  ..., -0.4899, -0.3066,  0.0675],
         [ 0.0290, -1.3258,  0.3085,  ..., -0.5257,  0.9768,  0.6652],
         [ 0.1073, -0.0619, -0.0716,  ..., -1.8726, -0.6527,  0.8814],
         ...,
         [-1.4692, -0.2195, -0.1264,  ..., -2.3614,  0.2694, -0.0432],
         [-1.4824, -0.2189, -0.1254,  ..., -2.3543,  0.2633, -0.0390],
         [-1.4006, -0.1564, -0.0655,  ..., -2.3871,  0.2966, -0.1056]]],
       device='cuda:0')
```

可以看到是有内容的，pdb后也是进入了else。

这里有两个模型输入，我需要分别知道他们是什么模型：

```python
control = self.control_model(x=x_noisy, hint=torch.cat(cond['c_concat'], 1), timesteps=t, context=cond_txt)
```

从这里的输入我看到了x_noisy，hint，timesteps，context。与老师提示的对上了，很可能是controlnet，我需要进一步证明

进一步调试：

```
> /root/miniconda/lib/python3.8/site-packages/torch/nn/modules/module.py(1613)__getattr__()
-> return modules[name]
(Pdb) l
1608                if name in _buffers:
1609                    return _buffers[name]
1610            if '_modules' in self.__dict__:
1611                modules = self.__dict__['_modules']
1612                if name in modules:
1613 ->                 return modules[name]
1614            raise AttributeError("'{}' object has no attribute '{}'".format(
1615                type(self).__name__, name))
1616
1617        def __setattr__(self, name: str, value: Union[Tensor, 'Module']) -> None:
1618            def remove_from(*dicts_or_sets):
```

到这里可以看到是通脱torch创建了一个modules，这里打印返回值modlues[name]，出现：

```
(Pdb) p modules[name]
ControlNet(
  (time_embed): Sequential(
    (0): Linear(in_features=320, out_features=1280, bias=True)
    (1): SiLU()
    (2): Linear(in_features=1280, out_features=1280, bias=True)
  )
  (input_blocks): ModuleList(
    (0): TimestepEmbedSequential(
      (0): Conv2d(4, 320, kernel_size=(3, 3), stride=(1, 1),
      ...
```

可以看到确实是Controlnet，返回，打印输入：

```
(Pdb) p x_noisy.shape
torch.Size([1, 4, 32, 48])
(Pdb) p t.shape
torch.Size([1])
(Pdb) p cond_txt.shape
torch.Size([1, 77, 768])
(Pdb) p torch.cat(cond['c_concat'], 1).shape
torch.Size([1, 3, 256, 384])
```

确定了Controlnet输入的shape：

```
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)
hint = torch.randn(1, 3, 256, 384, dtype=torch.float32)

input_names = ["x_noisy", "hint", "timestep", "context"] #
```

> 输入方面Input_name顺序这里不知道有没有规则，我的判断应该是在torch.onnx.export()函数中，input_names要与输入位置一致。
>
> 答：input_names只是给输入参数命名，一一对应即可

> 输出方面，老师提示中是"latent"，我并未找到其出处，可能起名没有什么限制。
>
> 答：就是要保证这个名称在 unet 输出的定义和 decoder 输入名称一致就行

```
output_names = ["latent"]
```

仿照clip转换的代码，还需要：

```
onnx_path = "./onnx/CONTROL_NET.onnx"  # 设置onnx输出路径

#  模型导出
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
    )

# 验证onnx模型
output = control_net(x_noisy, hint, timestep, context)  # 得到模型输出结果
input_dicts = {"x_noisy": x_noisy.numpy(), "hint": hint.numpy(), "timestep": timestep.numpy(), "timestep": timestep.numpy()}  # onnxruntime推理输入字典
onnxruntime_check(onnx_path, input_dicts, [output])  # 这个函数我看了下是为了检测onnx模型导出是否正确
```

其中关于模型导出有几个问题：

- dynamic_axes=dynamic_axes是输入输出动态维度，我看模型controlnet输入输出应该都是固定的，我这个不确定

> clip模型基于文本对图像分类，也可以基于图像对文本分类，是SD的文本条件约束。这个输入肯定是动态大小输入，主要是文本条数。
>
> 看其导出参数设置：
>
> ```
> dynamic_axes = {
>  "input_ids": {1: "S"},  # 输入 "input_ids" 的第 1 维度是动态的，命名为 "S"
>  "last_hidden_state": {1: "S"}  # 输出 "last_hidden_state" 的第 1 维度是动态的，命名为 "S"
> }
> ```
>
> - `"S"` 是该动态维度的名称（可以自定义，通常使用有意义的名称，如 `"sequence_length"`）。
>
> 因此在torch.onnx.export内添加参数即可

- keep_initializers_as_inputs=True这个参数是看老师视频的答案里面有，但是不知道为什么这个要设置为True

> torch.onnx.export()函数的keep_initializers_as_inputs参数，老师说设置成 默认值 None 大多数时候都是没问题的，拿 none false true 测试 三者导出的 onnx 都能正常用。
>
> 以forward(a, b=torch.Tensor([1]))为例
>
> - `keep_initializers_as_inputs=False`：带有默认值的参数（如 `b=torch.Tensor([1])`）会被视作一个常量（initializer），这个常量会在导出到 ONNX 时被嵌入到模型中。因此，当你调用导出的 ONNX 模型时，只需要传入 `a`，`b` 会自动使用其默认值。并且**不能**在调用时改变 `b`，除非你修改模型的定义，让 `b` 成为一个显式的输入。如果你想要动态地给 `b` 传入不同的值，你需要将 `keep_initializers_as_inputs` 设置为 `True`，这样 `b` 就会作为一个输入（input）被处理，而不是作为一个默认值。也就是说只能以forward(a)形式调用，forward(a, b)不行
> - `keep_initializers_as_inputs=True`：默认值被视为模型输入，必须显式传递给模型。

- forwad什么时候该替换？

> 老师在课程里面讲到，onnx模型ONNX在实践中主要支持张量输入，要将非张量类型传给onnx最好需要将对象转成张量，如文本要通过嵌入层转换为张量，而图片数据可直接作为张量输入。

因此control_net的pytorch转onnx代码最终为：

```python
def export_control_net_model():
    control_net = hk.model.control_model

    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)
    hint = torch.randn(1, 3, 256, 384, dtype=torch.float32)

    onnx_path = "./onnx/CONTROL_NET.onnx"

    input_names = ["x_noisy", "hint", "timestep", "context"] # 这里不知道输入顺序是否要与代码保持一致，干脆与原函数输入顺序一样
    output_names = ["latent"]

    # 模型输出
    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("====================== CONTROL_NET model export onnx done!")

    # 验证onnx模型
    output = 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, output)
    print("======================= CONTROL_NET onnx model verify done!")
```

运行export_onnx.py:

```
======================= CONTROL_NET onnx model verify done!
root@6bbe6b07716d:~/TensorRT-StableDiffusion# cd onnx
root@6bbe6b07716d:~/TensorRT-StableDiffusion/onnx# ls
CLIP.onnx  CONTROL_NET.onnx  readme.md
root@6bbe6b07716d:~/TensorRT-StableDiffusion/onnx#
```

模型导出成功~

> 错误补充1：
>
> ```
> Traceback (most recent call last):
>   File "export_onnx_full.py", line 215, in <module>
>     main()
>   File "export_onnx_full.py", line 210, in main
>     export_control_net_model()
>   File "export_onnx_full.py", line 147, in export_control_net_model
>     onnxruntime_check(onnx_path, input_dicts, [output])
>   File "export_onnx_full.py", line 49, in onnxruntime_check
>     ret = np.allclose(result[i], torch_outputs[i].detach().numpy(), rtol=1e-03, atol=1e-05, equal_nan=False)
> AttributeError: 'list' object has no attribute 'detach'
> ```
>
> 对应代码是

```
onnxruntime_check(onnx_path, input_dicts, [output])
```

因为controlnet结果已经是list了，所以不用加output不用加[]

错误补充2：

```
2025-02-12 19:39:32.362940235 [W:onnxruntime:, graph.cc:1312 Graph] Initializer onnx::MatMul_4410 appears in graph inputs and will not be treated as
 constant value/weight. This may prevent some of the graph optimizations, like const folding. Move it out of graph inputs if there is no need to ove
rride it, by either re-generating the model with latest exporter/converter or with the tool onnxruntime/tools/python/remove_initializer_from_input.p
y.
2025-02-12 19:39:32.362945135 [W:onnxruntime:, graph.cc:1312 Graph] Initializer onnx::MatMul_4431 appears in graph inputs and will not be treated as
 constant value/weight. This may prevent some of the graph optimizations, like const folding. Move it out of graph inputs if there is no need to ove
rride it, by either re-generating the model with latest exporter/converter or with the tool onnxruntime/tools/python/remove_initializer_from_input.p
y.
2025-02-12 19:39:32.362949844 [W:onnxruntime:, graph.cc:1312 Graph] Initializer onnx::MatMul_4432 appears in graph inputs and will not be treated as
 constant value/weight. This may prevent some of the graph optimizations, like const folding. Move it out of graph inputs if there is no need to ove
rride it, by either re-generating the model with latest exporter/converter or with the tool onnxruntime/tools/python/remove_initializer_from_input.p
y.
2025-02-12 19:39:32.362957250 [W:onnxruntime:, graph.cc:1312 Graph] Initializer onnx::MatMul_4433 appears in graph inputs and will not be treated as
 constant value/weight. This may prevent some of the graph optimizations, like const folding. Move it out of graph inputs if there is no need to ove
rride it, by either re-generating the model with latest exporter/converter or with the tool onnxruntime/tools/python/remove_initializer_from_input.p
y.
2025-02-12 19:39:32.362963124 [W:onnxruntime:, graph.cc:1312 Graph] Initializer onnx::Mul_4434 appears in graph inputs and will not be treated as co
nstant value/weight. This may prevent some of the graph optimizations, like const folding. Move it out of graph inputs if there is no need to overri
de it, by either re-generating the model with latest exporter/converter or with the tool onnxruntime/tools/python/remove_initializer_from_input.py.
2025-02-12 19:39:32.362968726 [W:onnxruntime:, graph.cc:1312 Graph] Initializer onnx::Add_4435 appears in graph inputs and will not be treated as co
nstant value/weight. This may prevent some of the graph optimizations, like const folding. Move it out of graph inputs if there is no need to overri
de it, by either re-generating the model with latest exporter/converter or with the tool onnxruntime/tools/python/remove_initializer_from_input.py.
2025-02-12 19:39:32.362973714 [W:onnxruntime:, graph.cc:1312 Graph] Initializer onnx::Mul_4436 appears in graph inputs and will not be treated as co
nstant value/weight. This may prevent some of the graph optimizations, like const folding. Move it out of graph inputs if there is no need to overri
de it, by either re-generating the model with latest exporter/converter or with the tool onnxruntime/tools/python/remove_initializer_from_input.py.
2025-02-12 19:39:32.362978655 [W:onnxruntime:, graph.cc:1312 Graph] Initializer onnx::Add_4437 appears in graph inputs and will not be treated as co
nstant value/weight. This may prevent some of the graph optimizations, like const folding. Move it out of graph inputs if there is no need to overri
de it, by either re-generating the model with latest exporter/converter or with the tool onnxruntime/tools/python/remove_initializer_from_input.py.
Error onnxruntime_check
Error onnxruntime_check
Error onnxruntime_check
Error onnxruntime_check
======================= CONTROL_NET onnx model verify done!
root@3f2b543c614d:~/TensorRT-StableDiffusion#
```

黄色部分是warning，没有关系

onnxruntime_check函数报错，看看是怎么检查模型的

```
def onnxruntime_check(onnx_path, input_dicts, torch_outputs):
    onnx_model = onnx.load(onnx_path)
    # onnx.checker.check_model(onnx_model)
    sess = rt.InferenceSession(onnx_path)
    # outputs = self.get_output_names()
    # latent input
    # data = np.zeros((4, 77), dtype=np.int32)
    result = sess.run(None, input_dicts)

    for i in range(0, len(torch_outputs)):
        ret = np.allclose(result[i], torch_outputs[i].detach().numpy(), rtol=1e-03, atol=1e-05, equal_nan=False)
        if ret is False:
            print("Error onnxruntime_check")
            # import pdb; pdb.set_trace()
```

可见是将onnx文件导入，用onnxruntime运行得到结果与输入结果进行对比。

np.allclose是Numpy中的一个函数，用于判断两个数组是否在数值上近似相等。

定义：

```
numpy.allclose(a, b, rtol=1e-5, atol=1e-8, equal_nan=False)
```

- a, b: 输入的两个数组，需要比较它们是否相等。
- rtol: 相对误差的容忍度（默认值是1×10^{-5}）。计算方式是基于两个数组对应元素的大小。
- atol: 绝对误差的容忍度（默认值是1×10^{-8}）。计算方式是一个全局的固定误差容忍度。
- equal_nan: 是否将两个 NaN 视为相等。默认值是 False（即两个 NaN 不被视为相等）。设置为 True 后，两个对应位置的 NaN 将被视为相等

判断标准：

> 两个数组对应元素a和b满足以下条件时，视为近似相等：
>
> $$|a-b| \leq atol + rtol \cdot |b|$$
>
> 转onnx运行后，输出与原值有误差是正常现象，重要的是能接受的误差范围是多少。

ControlUnet

继续看下一步：

```
eps = diffusion_model(x=x_noisy, timesteps=t, context=cond_txt,
control=control, only_mid_control=self.only_mid_control)
```

打印查看diffusion_model，可以知道这就是controlunet：

```
ControlledUnetModel(
  (time_embed): Sequential(
    (0): Linear(in_features=320, out_features=1280, bias=True)
    (1): SiLU()
    (2): Linear(in_features=1280, out_features=1280, bias=True)
  )
  (input_blocks): ModuleList(
    (0): TimestepEmbedSequential(
      (0): Conv2d(4, 320, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    )
      ...
```

从入口可以看出输入有4个：x_noisy，timesteps，context，control。其中，x_noisy，timesteps，context的shape和controlnet的一样，而control是controlnet的输出。打印一下：

```
(Pdb) for i, tensor in enumerate(control): print(f"Tensor {i}: {tensor.shape}")
Tensor 0: torch.Size([1, 320, 32, 48])
Tensor 1: torch.Size([1, 320, 32, 48])
Tensor 2: torch.Size([1, 320, 32, 48])
Tensor 3: torch.Size([1, 320, 16, 24])
Tensor 4: torch.Size([1, 640, 16, 24])
Tensor 5: torch.Size([1, 640, 16, 24])
Tensor 6: torch.Size([1, 640, 8, 12])
Tensor 7: torch.Size([1, 1280, 8, 12])
Tensor 8: torch.Size([1, 1280, 8, 12])
Tensor 9: torch.Size([1, 1280, 4, 6])
Tensor 10: torch.Size([1, 1280, 4, 6])
Tensor 11: torch.Size([1, 1280, 4, 6])
Tensor 12: torch.Size([1, 1280, 4, 6])
```

因此，controlunet输如可以设为：

```
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 = [
    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),
]

input_names = ["x_noisy", "timestep", "context"]
for i in range(0, len(control)):
    input_names.append("control" + str(i))
```

输出

```
output_names = ["latent"]

onnx_path = "./onnx/CONTROL_UNET.onnx"

torch.onnx.export(
    controlled_unet_model,
    (x_noisy, timestep, context, control),
    onnx_path,
    verbase=True,
    opset_version=18,
    do_constant_folding=True,
    input_names=input_names,
    output_names=output_names
)
```

验证

```
output = controlled_unet_mdoel(x_noisy, hint, timestep, control)
input_dicts = {"x_noisy": x_noisy.numpy(), "timestep": timestep.numpy(),
"context": context.numpy(), "control": control.numpy()}
onnxruntime_check(onnx_path, input_dicts, output)
```

总结:

```
def export_controlled_unet_model():
    controlled_unet_mdoel = hk.model.model.diffusion_model

    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 = [
        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),
```

```python
            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),
        ]
        # import pdb; pdb.set_trace()
        onnx_path = "./onnx/CONTROL_UNET.onnx"

        input_names = ["x_noisy", "timestep", "context"]
        for i in range(0, len(control)):
            input_names.append("control" + str(i))
        output_names = ["latent"]

        # 模型输出
        torch.onnx.export(
            controlled_unet_mdoel,
            (x_noisy, timestep, context, control),
            onnx_path,
            verbose=True,
            opset_version=18,
            do_constant_folding=True,
            input_names=input_names,
            output_names=output_names,
        )
        print("====================== CONTROL_UNET model export onnx done!")

        # 验证onnx模型
        # 这里如果先计算output会导致control清空因此要先写input_dicts,不然for循环执行不了(len(control)=0)
        input_dicts = {"x_noisy": x_noisy.numpy(), "timestep": timestep.numpy(), "context": context.numpy()}
        for i in range(0, len(control)):
            input_dicts["control" + str(i)]= control[i].numpy()

        output = controlled_unet_mdoel(x_noisy, timestep, context, control)


        onnxruntime_check(onnx_path, input_dicts, output)
        print("======================= CONTROL_UNET onnx model verify done!")
```

> 报错问题：
>
> ```
> ====================== CONTROL_UNET model export onnx done!
> Traceback (most recent call last):
>   File "export_onnx_full.py", line 219, in <module>
>     main()
>   File "export_onnx_full.py", line 215, in main
>     export_controlled_unet_model()
>   File "export_onnx_full.py", line 203, in export_controlled_unet_model
>     onnxruntime_check(onnx_path, input_dicts, output)
>   File "export_onnx_full.py", line 46, in onnxruntime_check
>     result = sess.run(None, input_dicts)
>   File "/root/miniconda/lib/python3.8/site-packages/onnxruntime/capi/onnxruntime_inference_collection.py", line 216, in run
>     self._validate_input(list(input_feed.keys()))
>   File "/root/miniconda/lib/python3.8/site-packages/onnxruntime/capi/onnxruntime_inference_collection.py", line 198, in _validate_input
>     raise ValueError(
> ValueError: Required inputs (['control0', 'control1', 'control2', 'control3', 'control4', 'control5', 'control6', 'control7', 'control8', 'control9', 'control10', 'control11', 'control12']) are missing from input feed (['x_noisy', 'timestep', 'context']).
> ```
>
> 显示是在验证模型时，输入没有control选项。
>
> 但是我明明写了
>
> ```python
> input_dicts = {"x_noisy": x_noisy.numpy(), "timestep": timestep.numpy(), "context": context.numpy()}
> for i in range(0, len(control)):
>  input_dicts["control" + str(i)]= control[i].numpy()
> ```

> input_dicts里应该有control才对。于是我pdb一下，发现input_dicts里没有control!
>
> 继续pdb，发现`for i in range(0, len(control)):`没有执行，查看len(control)=1，再查看control为空。但是在创建模型是control有内容。于是想到可能是在计算output结果时将control传入导致里面数据清空，pdb证实我的猜想是正确的，因此input_dicts要在output计算前
>
> 即：
>
> ```
> input_dicts = {"x_noisy": x_noisy.numpy(), "timestep": timestep.numpy(), "context": context.numpy()}
>  for i in range(0, len(control)):
>      input_dicts["control" + str(i)]= control[i].numpy()
>
> output = controlled_unet_mdoel(x_noisy, timestep, context, control)
> ```

decoder

之后回到canny2imageTRT.py

```
x_samples = self.model.decode_first_stage(samples)
x_samples = (einops.rearrange(x_samples, 'b c h w -> b h w c') * 127.5 + 127.5).cpu().numpy().clip(0, 255).astype(np.uint8)

results = [x_samples[i] for i in range(num_samples)]
```

这里的model是：

```
self.model = create_model('./models/cldm_v15.yaml').cpu()
```

pdb查看：

```
(Pdb) p decode_model
AutoencoderKL(
  (encoder): Encoder(
    (conv_in): Conv2d(3, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (down): ModuleList(
      (0): Module(
        (block): ModuleList(
          (0-1): 2 x ResnetBlock(
            (norm1): GroupNorm(32, 128, eps=1e-06, affine=True)
            (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    ...
```

AutoencoderKL是自动编码器，是一种数据压缩算法。

显然，根据`x_samples = self.model.decode_first_stage(samples)`知道，decoder输入只有一个samples。通过pdb，samples大小为[1, 4, 32, 48]

因此：

```python
def export_decoder_model():
    # control_net = hk.model.control_model

    decode_model = hk.model.first_stage_model
    decode_model.forward = decode_model.decode

    latent = torch.randn(1, 4, 32, 48, dtype=torch.float32)

    input_names = ["latent"]
    output_names = ["image"]

    onnx_path = "./onnx/DECODER.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 model export onnx done!")

    # 验证onnx模型
    output = decode_model(latent)
    input_dicts = {"latent": latent.numpy()}
    onnxruntime_check(onnx_path, input_dicts, [output])
    print("====================== DECODER onnx model verify done!")
```

模型测试

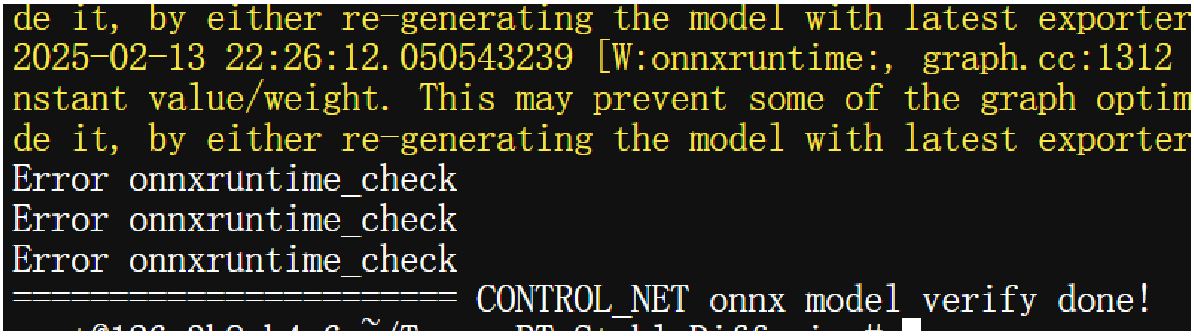
Controlunet和decoder的onnxruntime_check能在相对误差的容忍度rtol为1e-3、绝对误差容忍度atol为1e-5的情况下导出。

```
============ Diagnostic Run torch.onnx.export version 2.0.0+cu118 ============
verbose: False, log level: Level.ERROR
====================== 0 NONE 0 NOTE 0 WARNING 0 ERROR =======================

====================== CONTROL_UNET model export onnx done!
====================== CONTROL_UNET onnx model verify done!
root@126c3b8cb4e6:~/TensorRT-StableDiffusion#
```

```
stant value/weight. This may prevent some of the graph optimizatio
e it, by either re-generating the model with latest exporter/conve
2025-02-13 22:38:12.323682334 [W:onnxruntime:, graph.cc:1312 Graph
stant value/weight. This may prevent some of the graph optimizatio
e it, by either re-generating the model with latest exporter/conve
====================== DECODER onnx model verify done!
root@126c3b8cb4e6:~/TensorRT-StableDiffusion#
```

而Controlnet导出误差稍微大点，因此需要提高容忍度。



一般首先增加 `rtol`，可以让模型或者计算更加宽松地允许比例上的误差，这对于大范围的数值数据或具有较大数量级差异的数据很有用。然后，可以再考虑调整 `atol` 来容忍一些小的绝对差异，这样可以保证即使数值很小的差异也不被忽视。

测试：

- rtol=1e-3,atol=1e-4：通过