# STABLE DIFFUSION + CONTROLNET TENSORRT 性能优化

XUEWEI LI, NVIDIA DEVTECH, 2023.9

# STABLE DIFFUSION + CONTROLNET

Basic info

- Our goal is to optimize the ControlNet 1.0 model canny2image pipeline. (https://github.com/lllyasviel/ControlNet ).
- To limit the GPU memory usage smaller than 8G during inference, image resolution is 256 * 384.
- Target GPU is A10.
- We use PD score to compare the optimized pipeline generated image and the origin pytorch pipeline generated image. PD score computed as follow

```
block_idx = InceptionV3.BLOCK_INDEX_BY_DIM[2048]
model = InceptionV3([block_idx]).to("cuda")

def PD(base_img, new_img):
    inception_feature_ref, _ = fid_score.calculate_activation_statistics([base_img], model, batch_size = 1, device="cuda")
    inception_feature, _ = fid_score.calculate_activation_statistics([new_img], model, batch_size = 1, device="cuda")
    pd_value = np.linalg.norm(inception_feature - inception_feature_ref)
    pd_string = F"Perceptual distance to: {pd_value:.2f}"
    print(pd_string)
    return pd_value
```

- The closer the two images are, the lower their pd score will be.

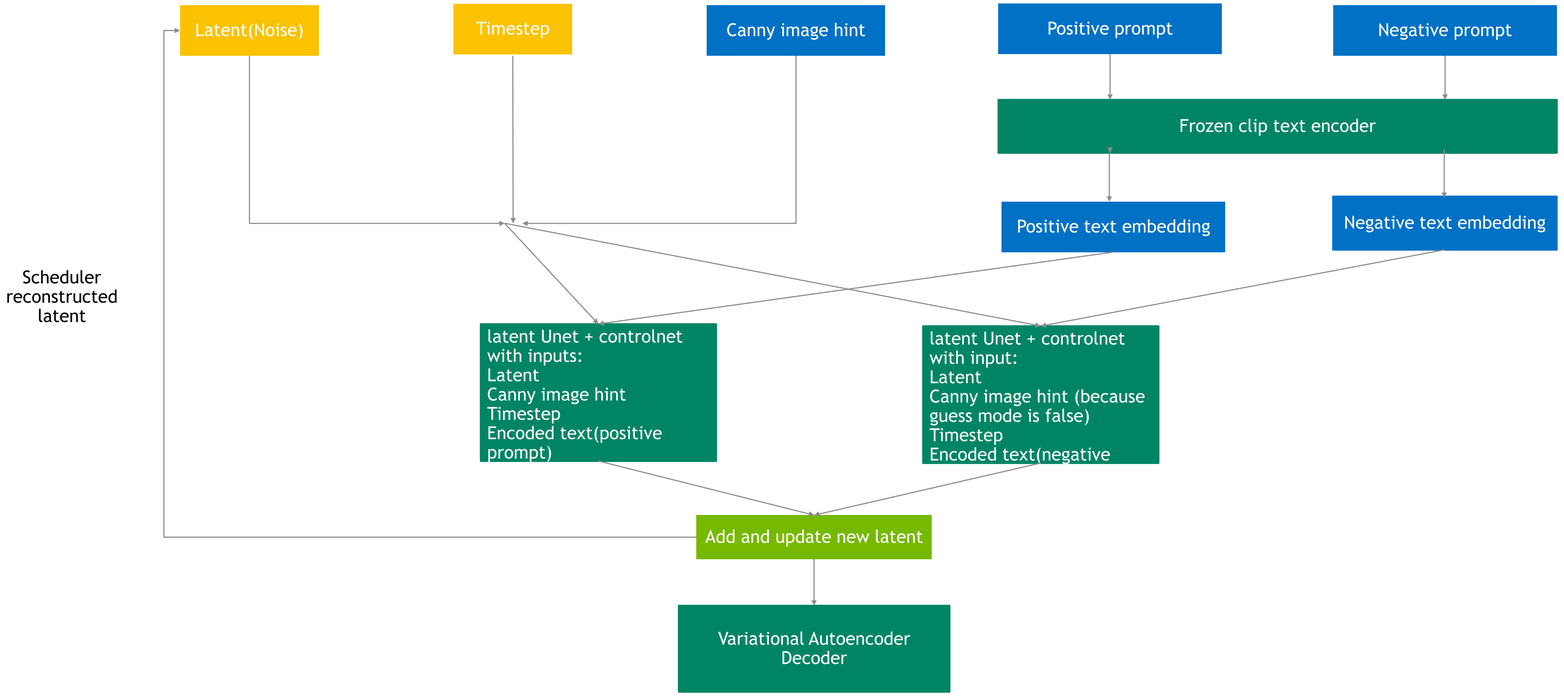
# STABLE DIFFUSION + CONTROLNET

scoring formula for this competition

- According to our experience:
  - Fp32 trt engine perceptual score is around 1.0.
  - Fp16 trt engine perceptual score is around 4.0.
  - Int8 trt engine perceptual score is around 8.0.
- We hope the int8 trt engine pipeline have the highest score. Therefore, when the perceptual score is less than 8, the players can see their scores increase significantly as the inference speed increases.
- What we didn’t well considered is that reducing the number of iterations can also accelerate inference while ensuring that the perceptual score is less than 8. The acceleration effect is even better than int8.
- Our scoring formula is not perfect then. Therefore, we revealed to all the players that they could directly reduce the number of iterations to accelerate the inference.
- I will use 20 reconstruction steps as an example to introduce the pipeline optimization method.

# STABLE DIFFUSION + CONTROLNET

## Pipeline structure

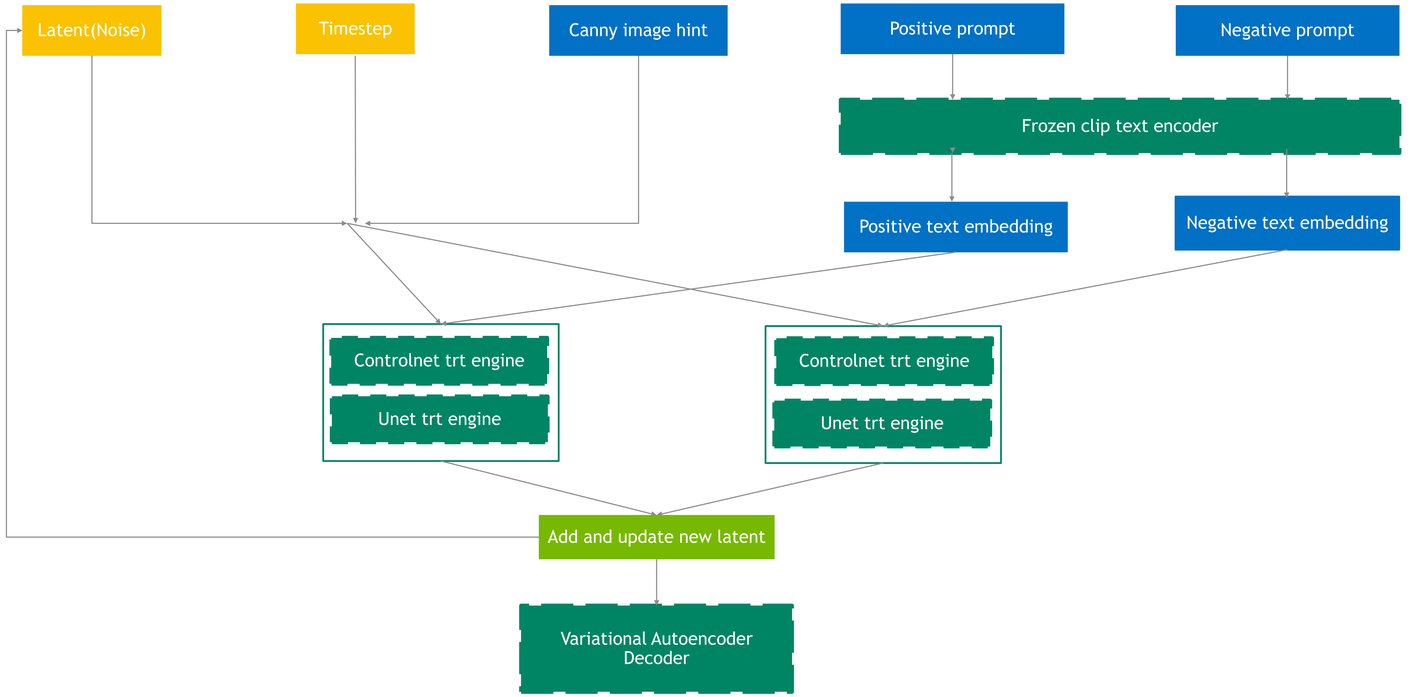



The pipeline inputs are: Latent, timestep, canny image hint, positive prompt and negative prompt. In the reconstruct loop, Latent and timestep will be updated every time (20 times in total in our case).
The pipeline includes models : Unet, clip and VAE. In the reconstruction loop, unet will be compute 20 times.

# STEP 1

## Export unet and controlnet to trt engine

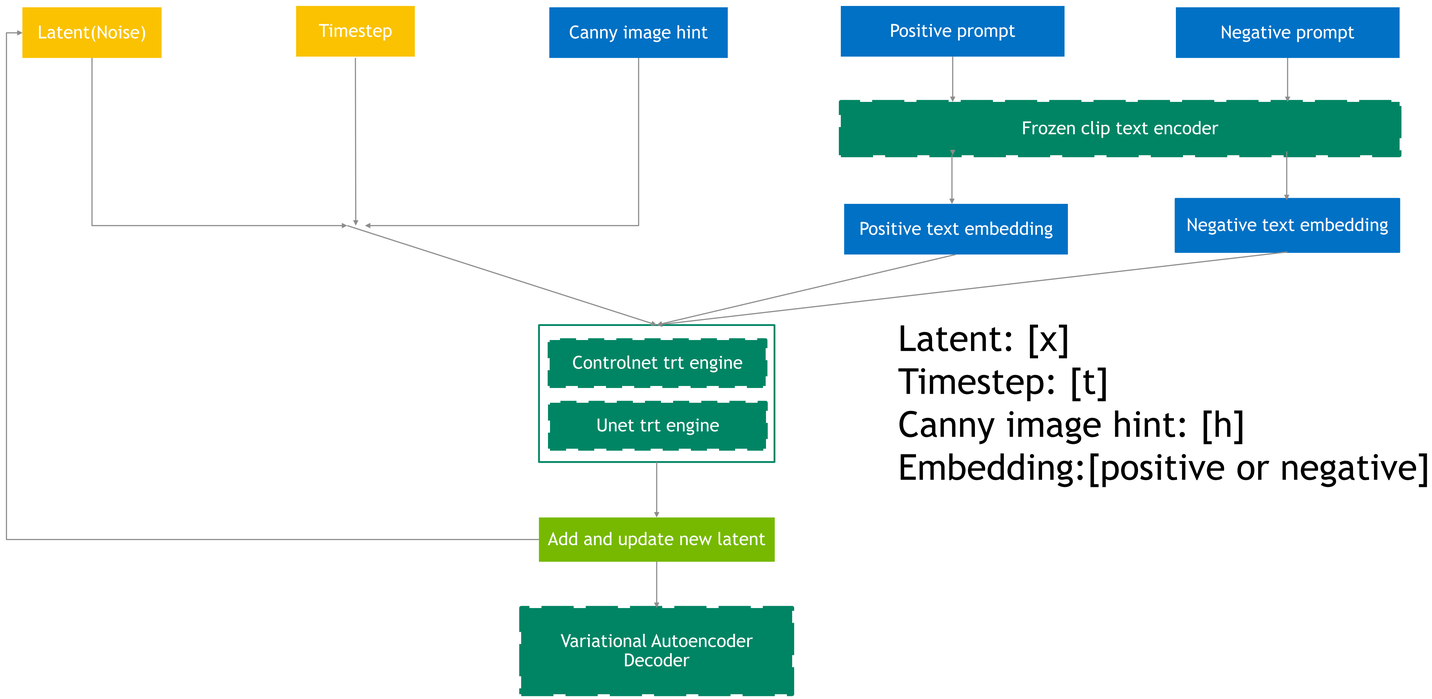
- Convert unet, controlnet, clip and VAE model to fp16 engine.
- Inference conditional and unconditional part separately.
- Clip model has fp32 negative infinite constant input. This will cause NaN outputs in fp16 precision trt engine. Just change them to fp16 negative large number can solve this issue.
- Inference time: 2600ms -> 634ms per image.



# STEP 2

Use batch size 2

- Batching always worth a try.
- Inference time: 634 -> 479.16ms per image

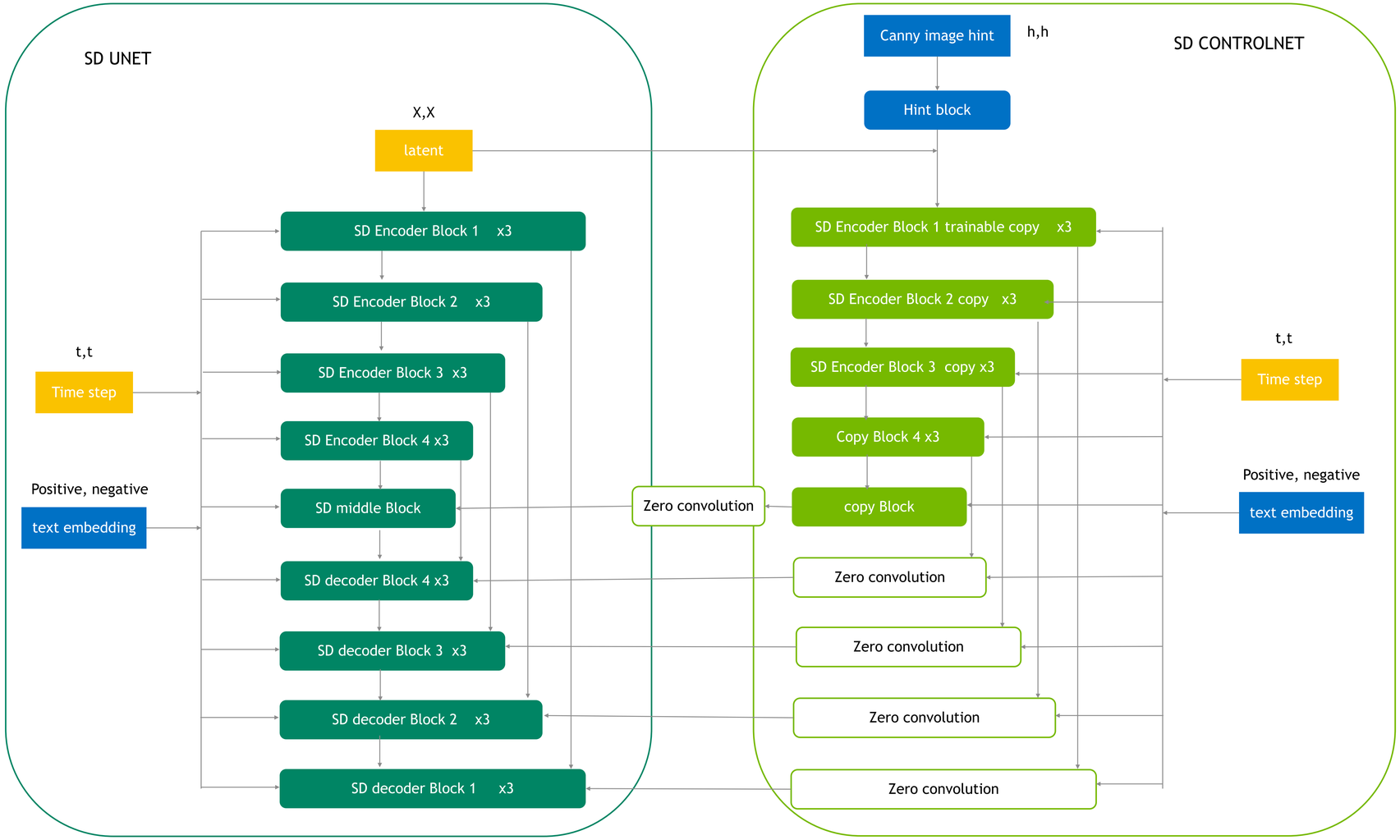



Latent: [x]
Timestep: [t]
Canny image hint: [h]
Embedding:[positive or negative]

Latent: [x,x]
Timestep: [t, t]
Canny image hint: [h, h]
Embedding:[positive, negative]

# STEP 3

## Remove duplicate compute in graph

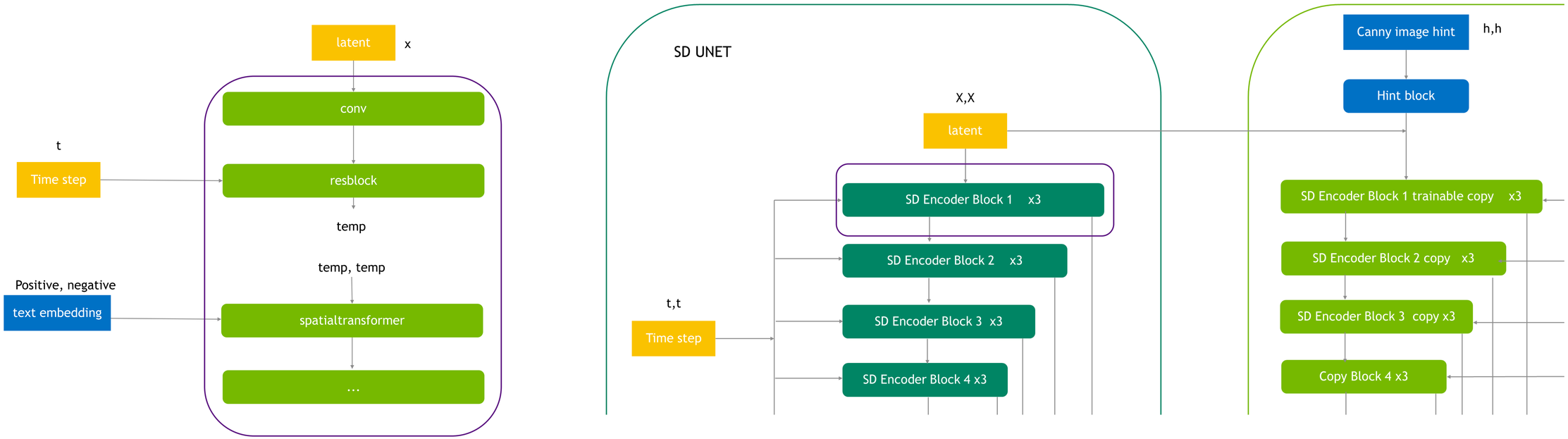



- For hint block: we only need to compute once, because the canny to image hint will not change during reconstruction.
- When we inference use batchsize = 2, timestep and latent are just duplicate itself, so we don’t need to duplicate them at first. We duplicate them when they meet text embedding.

# STEP 3

## Remove duplicate compute in graph

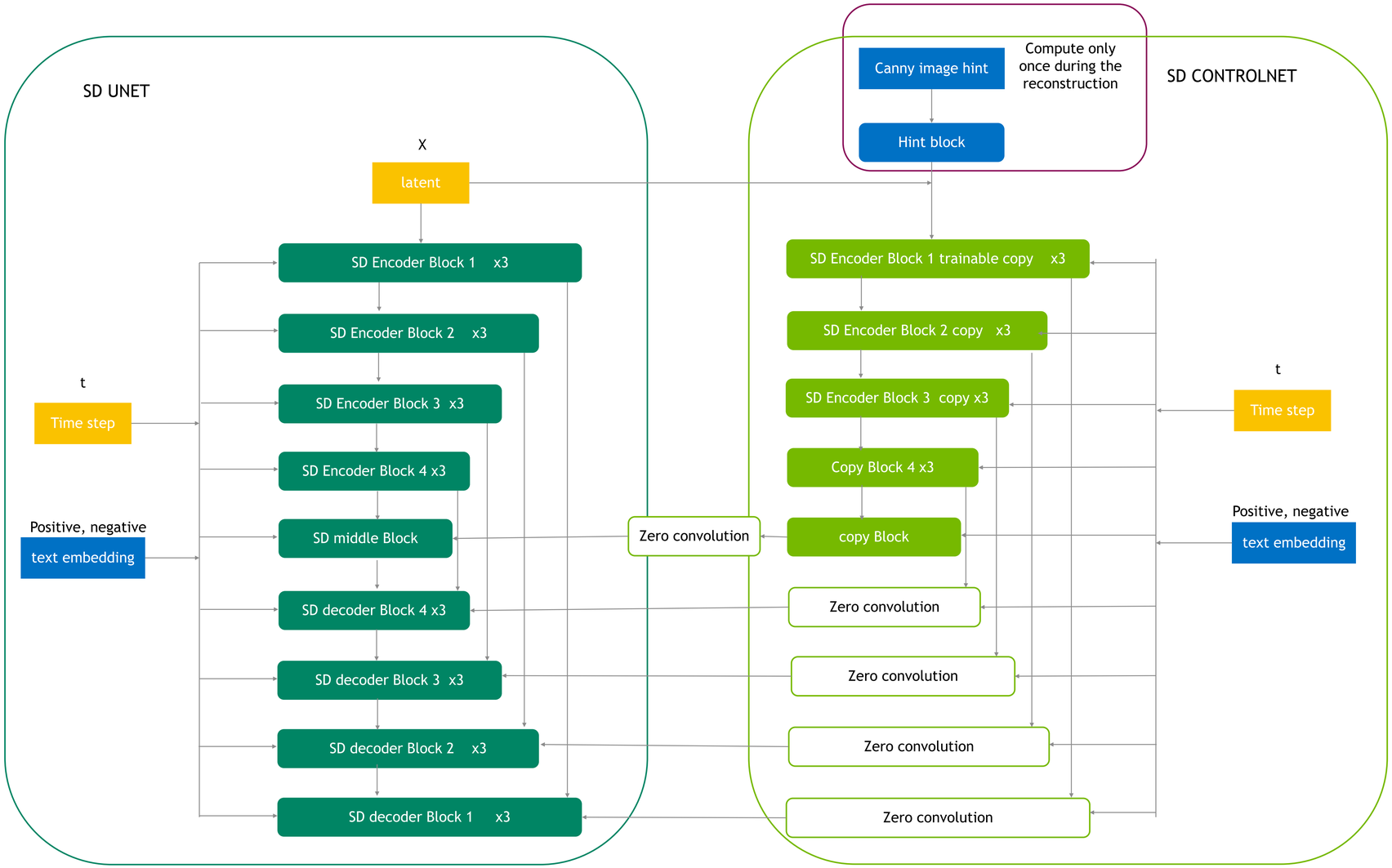
- For hint block: we only need to compute once, because the canny to image hint will not change during reconstruction.
- When we inference use batchsize = 2, timestep and latent are just duplicate itself, so we don’t need to duplicate them at first. We duplicate them when they meet text embedding.
- According to code, SD Encoder block 1 has node conv -> resblock -> spatialtransformer. Timestep and latent don’t need to duplicate before spatialtransformer.



# STEP 3

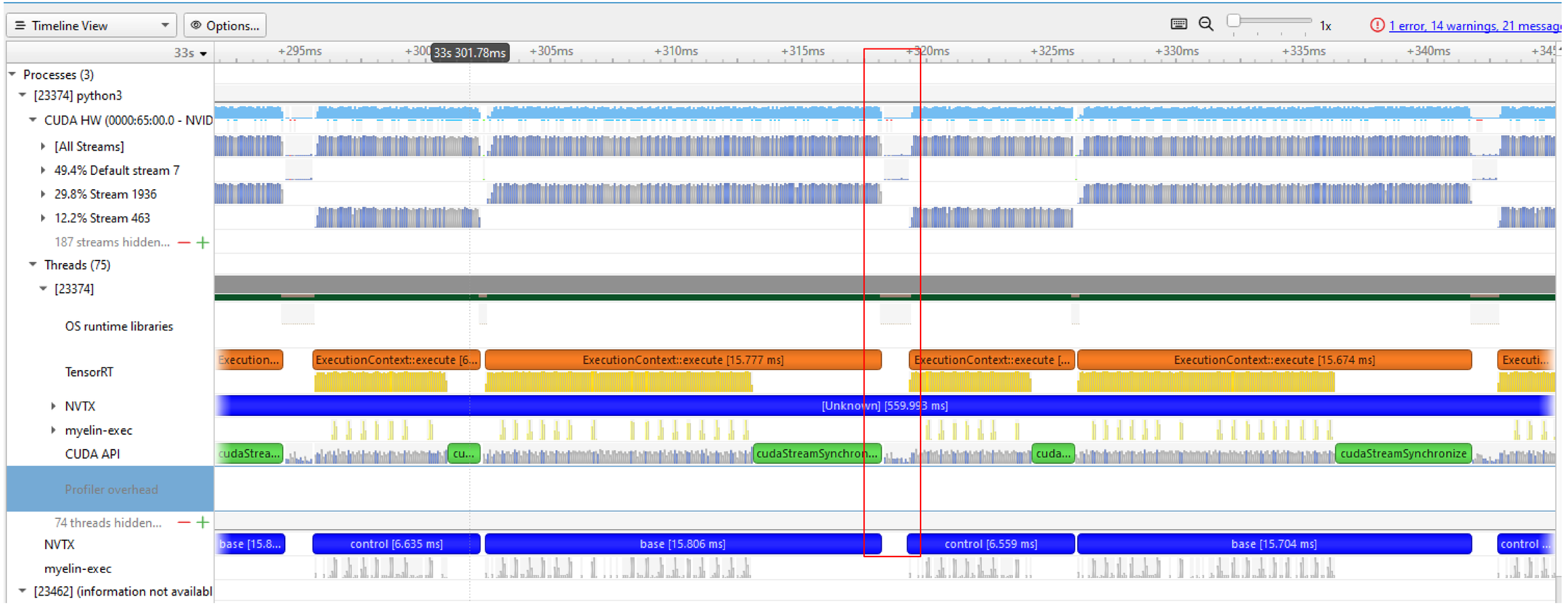
## Remove duplicate compute in graph

- inference time: 479ms -> 472ms per image



# STEP 4

Put the scheduler update logic in TRT.

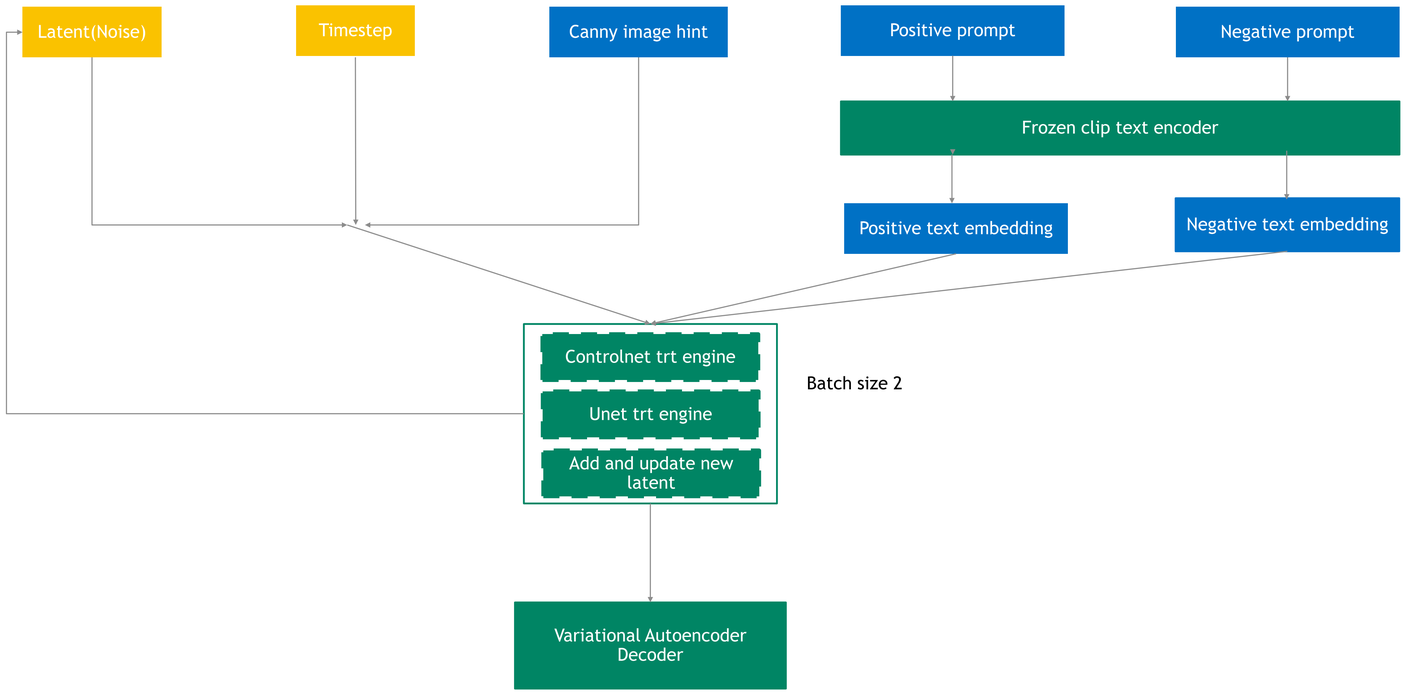



The GPU idle time is 1.12 ms

# STEP 4

Put the scheduler update logic in TRT.

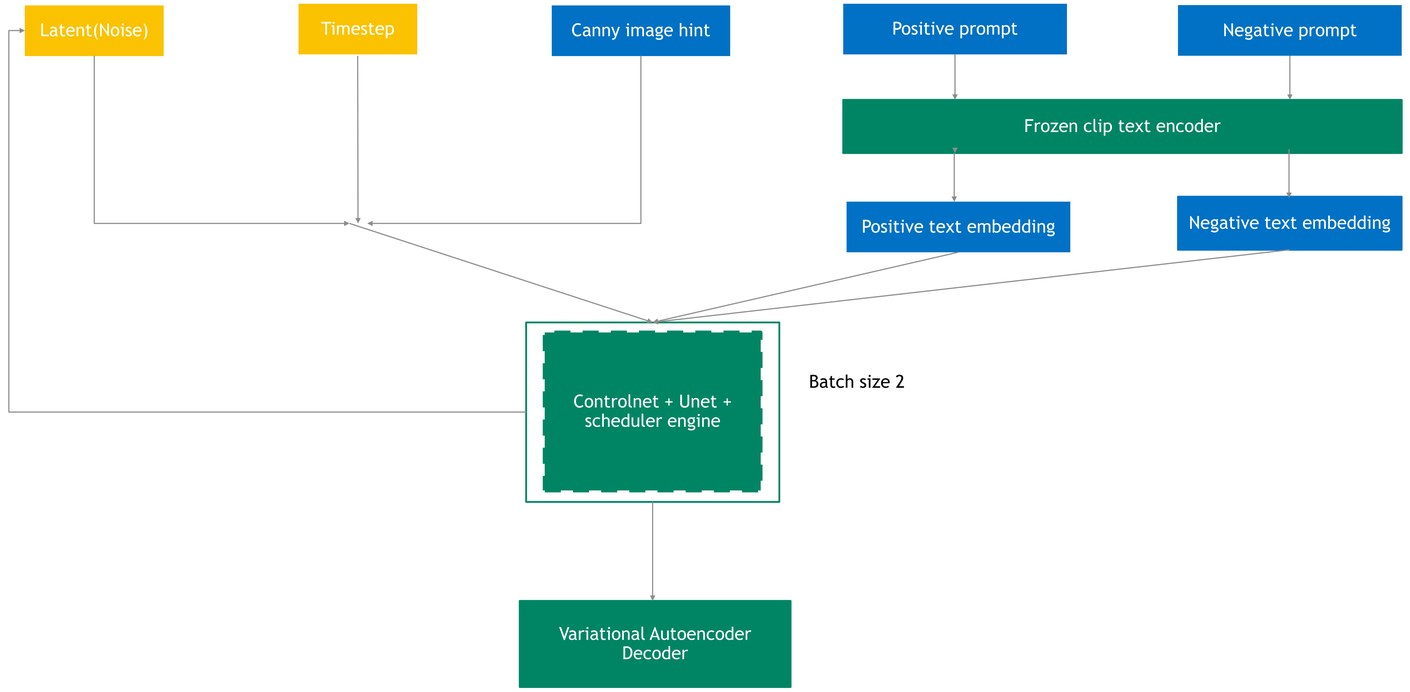
- Our code is adding the conditional and unconditional unet result and update latent in that gpu “spare” time.
- We can move that logic in trt to accelerate it.



# STEP 5

Fuse the Controlnet trt engine, Unet trt engine and the scheduler update logic in 1 engine.

- This can also help accelerate the inference, let discuss the reason later.
- Now the inference time is 472ms ->450ms.



# STEP 6

Cuda graph

- There are overheads associated with the submission of each operation to the GPU CUDA Graphs provide a mechanism to launch multiple GPU operations through a single CPU operation.
- We also can use cuda graph in TensorRT.
- When we use cuda graph, the engine input and output address should not be changed.

```
3   class cudagraph_engine():
4
5       def __init__(self, engine, context, tensor_list, stream):
6           # _, self.stream = cudart.cudaStreamCreate()
7           self.stream = stream
8           for i in range(len(tensor_list)):
9               name = engine.get_binding_name(i)
10              context.set_tensor_address(name, tensor_list[i].data_ptr())
11          context.execute_async_v3(self.stream)
12          # cudart.cudaStreamSynchronize(stream)
13          cudart.cudaStreamBeginCapture(self.stream, cudart.cudaStreamCaptureMode.cudaStreamCaptureModeGlobal)
14          context.execute_async_v3(self.stream)
15          _, graph = cudart.cudaStreamEndCapture(self.stream)
16          _, self.instance = cudart.cudaGraphInstantiate(graph, 0)
17
18
19      def infer(self):
20          cudart.cudaGraphLaunch(self.instance, self.stream)
```

# STEP 6

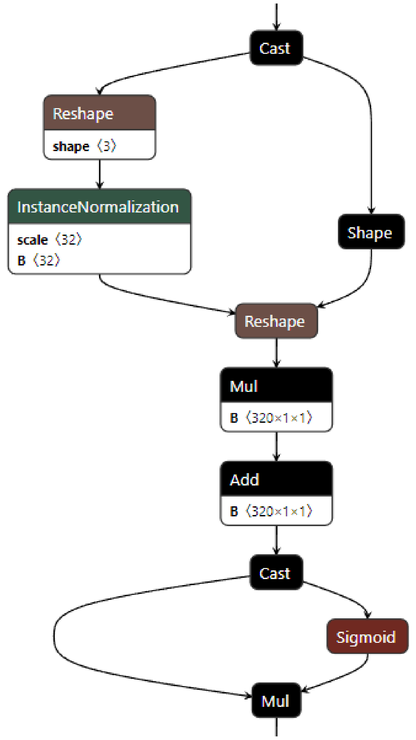
Do cuda graph work?

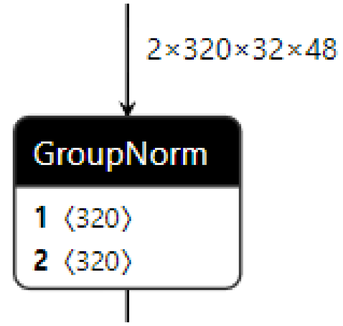
- Whether cudagraph can help accelerate our model can be tested by using trtexec “--useCudaGraph”.
- In our case, cudagraph can accelerate our model.
- With cudagraph, inference time is 450ms->425ms.
- By the way, “--cuda-graph-trace=node” need to be added when we profile a pipeline with cudagraph using nsys.


# STEP 7

Does plugin work?

- According to our player’s code. The GroupNorm plugin can accelerate the inference time.
- After adding the GroupNorm plugin, inference speed become 425ms->404ms.

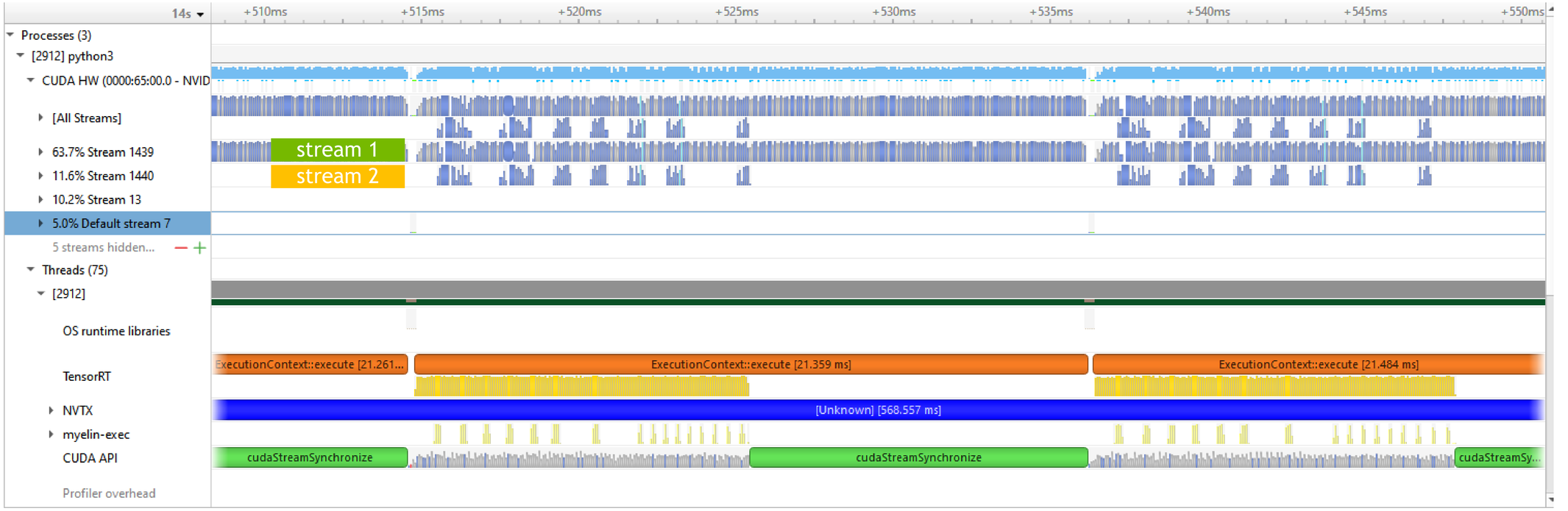


# STEP 8

Multi stream.

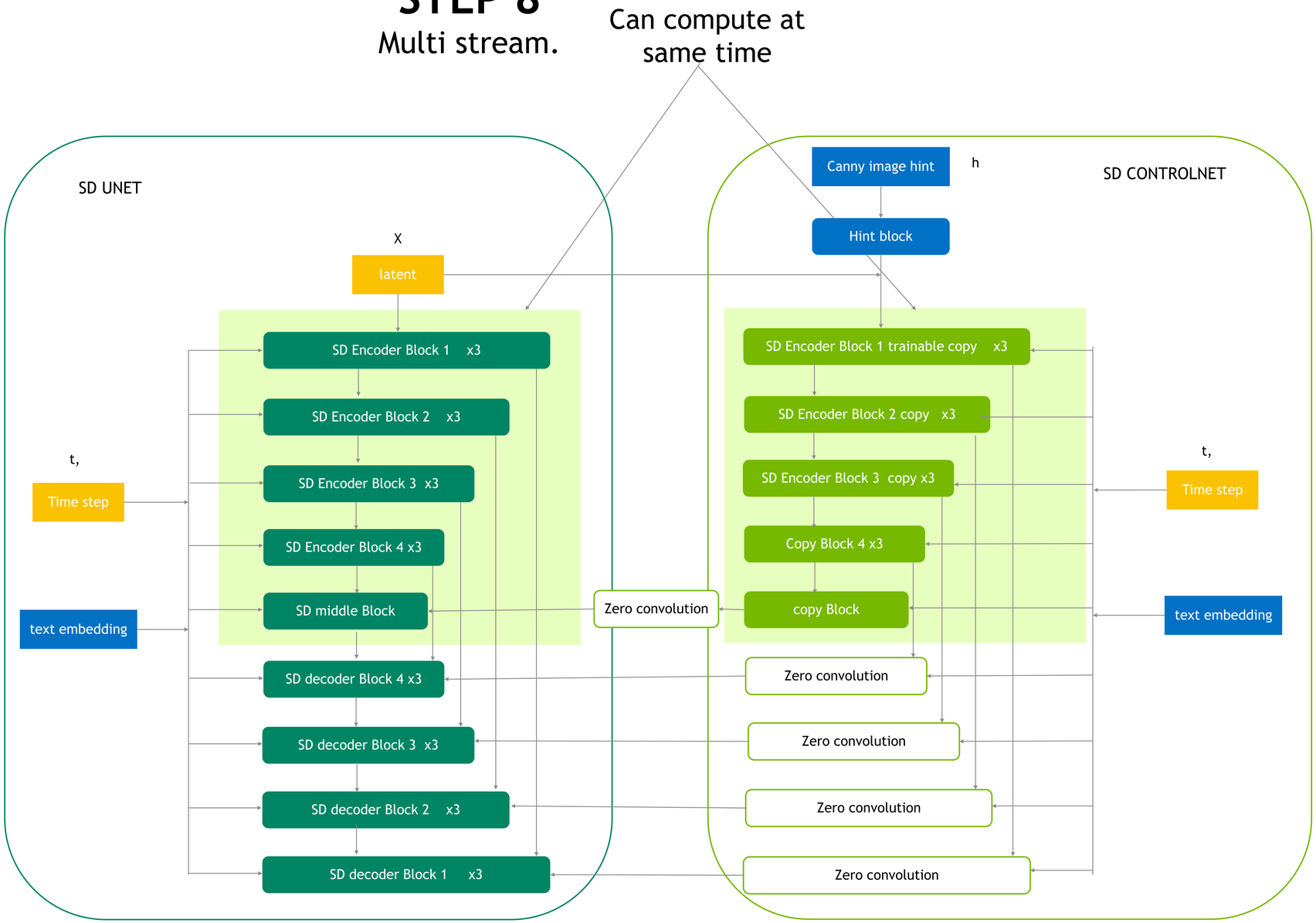
- Why step 5 (put all the compute in one engine can accelerate inference)?
  - One of the reasons is that TensorRT will try multi-stream during building engines when there are layers which can be computed at same time.
  - https://docs.nvidia.com/deeplearning/tensorrt/developer-guide/index.html#within-inference-multi-streaming
  - Let's look at the nsys timeline.



# STEP 8

Multi stream.

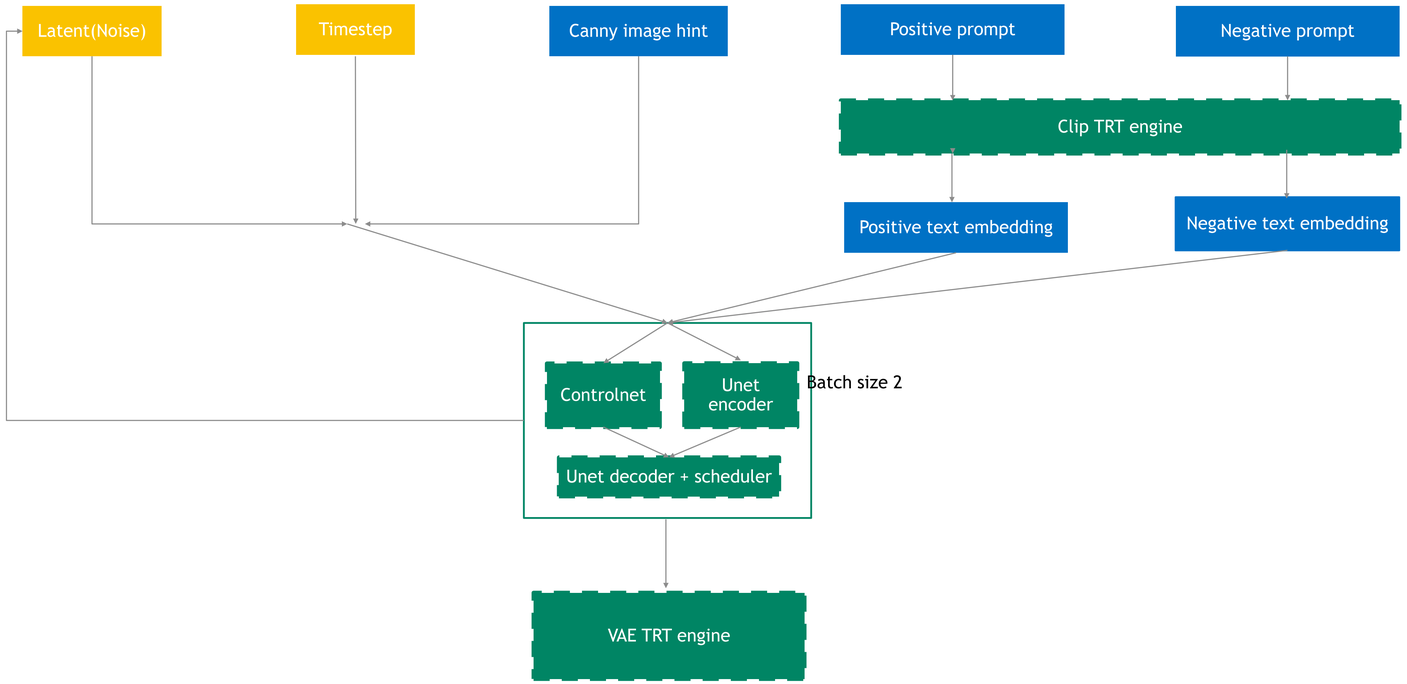
- Which layers can be computed at same time?
- SD UNET encoder and SD Controlnet have no dependency between each other.



# STEP 8

Multi stream.

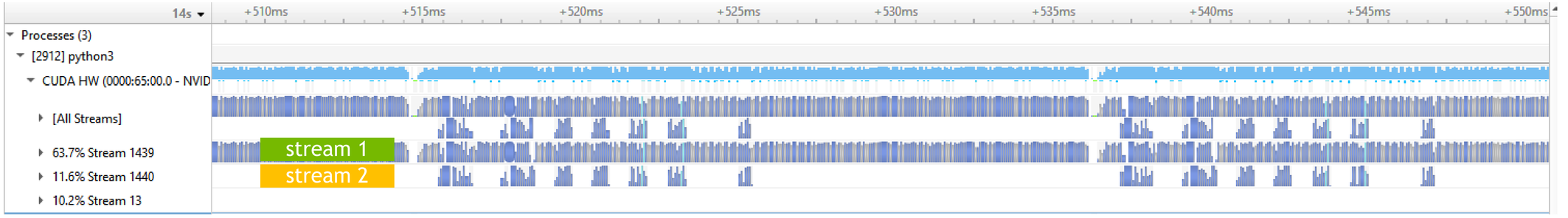
- We increase the parallelism by split the big one engine to 3 engines: constrolnet, SD UNET encoder and SD UNET decoder + scheduler logic. Then inference them on different stream.  Controlnet and SD UNET can be launched at same time.
- Inference time is: 404ms->393ms.

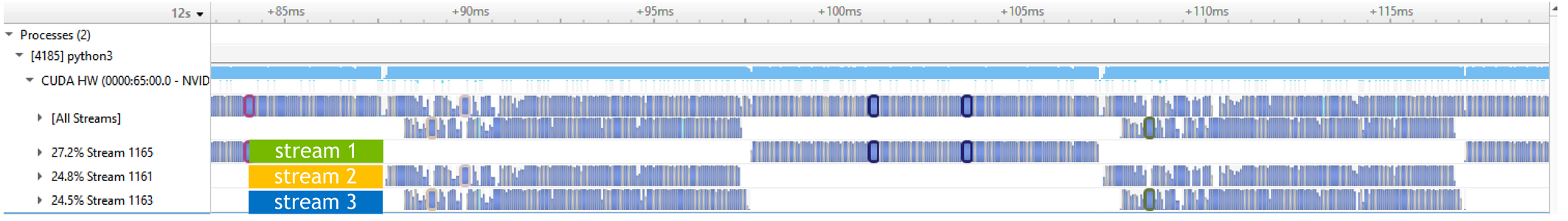


# STEP 8

Multi stream.

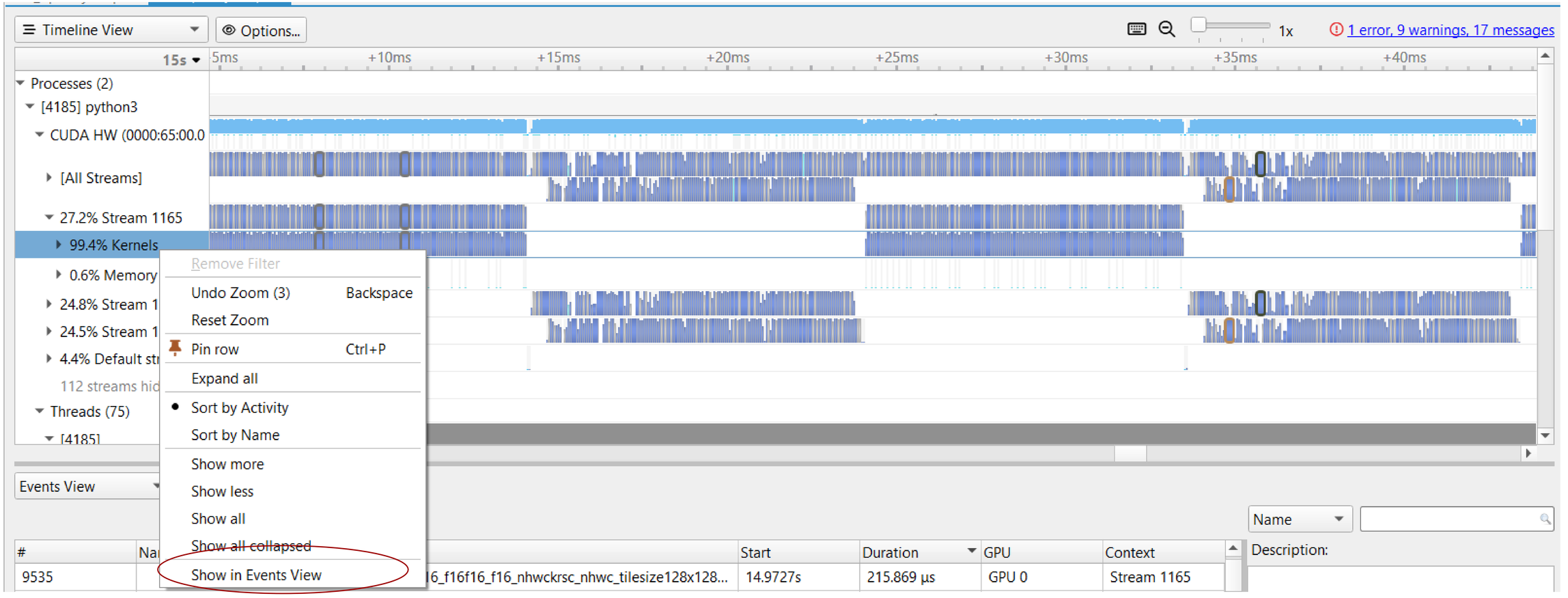
- We increase the parallelism by split one engine to three engine.



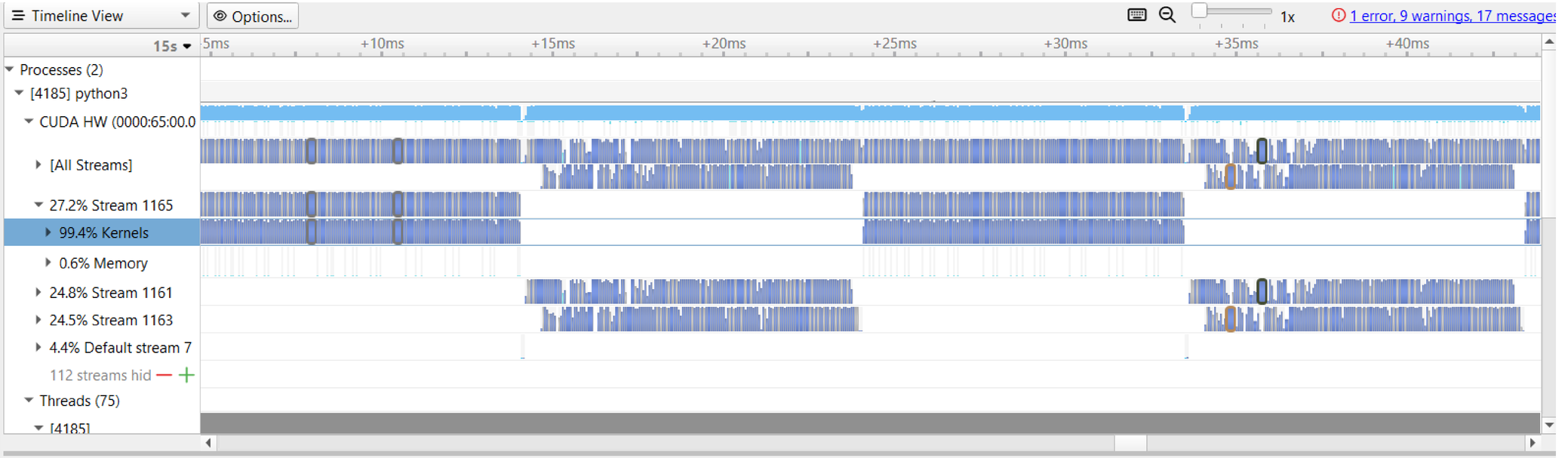
# STEP 9

Use TensorRT 9.1



# STEP 9

## Use TensorRT 9.1




Events View

Name

Description:

| # | Name | Start | Duration | GPU | Context |
|---|---|---|---|---|---|
| 9535 | sm80_xmma_fprop_implicit_gemm_f16f16_f16f16_f16_nhwckrsc_nhwc_tilesize128x128... | 14.9727s | 215.869 µs | GPU 0 | Stream 1165 |
| 715 | sm80_xmma_fprop_implicit_gemm_f16f16_f16f16_f16_nhwckrsc_nhwc_tilesize128x128... | 11.9876s | 215.614 µs | GPU 0 | Stream 1165 |
| 26335 | sm80_xmma_fprop_implicit_gemm_f16f16_f16f16_f16_nhwckrsc_nhwc_tilesize128x128... | 17.0756s | 213.630 µs | GPU 0 | Stream 1165 |
| 17815 | sm80_xmma_fprop_implicit_gemm_f16f16_f16f16_f16_nhwckrsc_nhwc_tilesize128x64x... | 16.0173s | 211.582 µs | GPU 0 | Stream 1165 |
| 17935 | sm80_xmma_fprop_implicit_gemm_f16f16_f16f16_f16_nhwckrsc_nhwc_tilesize128x128... | 16.0197s | 208.958 µs | GPU 0 | Stream 1165 |
| 13331 | mha_v2_1 | 15.1478s | 205.662 µs | GPU 0 | Stream 1165 |
| 18876 | mha_v2_1 | 16.061s | 205.630 µs | GPU 0 | Stream 1165 |
| 13371 | mha_v2_1 | 15.1489s | 205.373 µs | GPU 0 | Stream 1165 |
| 35211 | mha_v2_1 | 18.1491s | 205.341 µs | GPU 0 | Stream 1165 |
| 35676 | mha_v2_1 | 18.1699s | 204.542 µs | GPU 0 | Stream 1165 |
| 18791 | mha_v2_1 | 16.0588s | 204.350 µs | GPU 0 | Stream 1165 |
| 12531 | mha_v2_1 | 15.1095s | 204.286 µs | GPU 0 | Stream 1165 |

# STEP 9

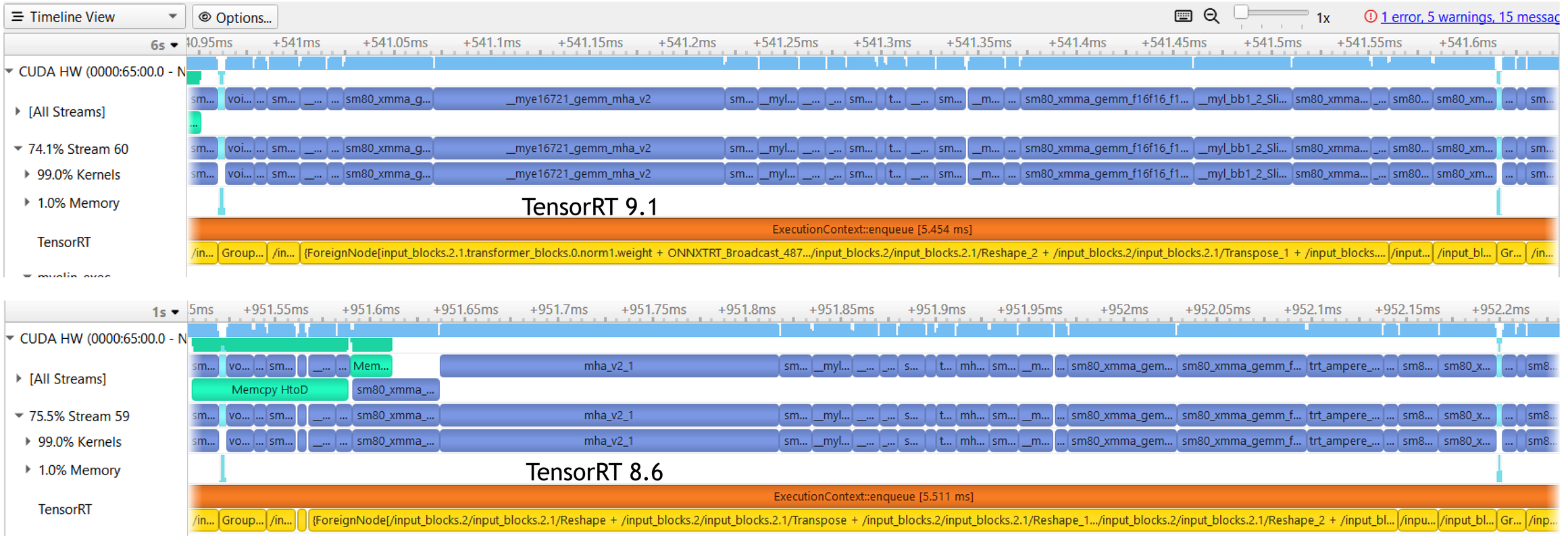
## Use TensorRT 9.1

| # | Name | Start | Duration | GPU | Context |
|---|---|---|---|---|---|
| 9955 | sm80_xmma_fprop_implicit_gemm_f16f16_f16f16_f16_nhwckrsc_nhwc_tilesize128x128... | 14.9915s | 225.661 µs | GPU 0 | Stream 1165 |
| 9415 | sm80_xmma_fprop_implicit_gemm_f16f16_f16f16_f16_nhwckrsc_nhwc_tilesize128x64x... | 14.9703s | 218.366 µs | GPU 0 | Stream 1165 |
| 1015 | sm80_xmma_fprop_implicit_gemm_f16f16_f16f16_f16_nhwckrsc_nhwc_tilesize128x64x... | 12.0038s | 218.365 µs | GPU 0 | Stream 1165 |
| 26215 | sm80_xmma_fprop_implicit_gemm_f16f16_f16f16_f16_nhwckrsc_nhwc_tilesize128x64x... | 17.0731s | 216.477 µs | GPU 0 | Stream 1165 |
| 9535 | sm80_xmma_fprop_implicit_gemm_f16f16_f16f16_f16_nhwckrsc_nhwc_tilesize128x128... | 14.9727s | 215.869 µs | GPU 0 | Stream 1165 |
| 715 | sm80_xmma_fprop_implicit_gemm_f16f16_f16f16_f16_nhwckrsc_nhwc_tilesize128x128... | 11.9876s | 215.614 µs | GPU 0 | Stream 1165 |
| 26335 | sm80_xmma_fprop_implicit_gemm_f16f16_f16f16_f16_nhwckrsc_nhwc_tilesize128x128... | 17.0756s | 213.630 µs | GPU 0 | Stream 1165 |
| 17815 | sm80_xmma_fprop_implicit_gemm_f16f16_f16f16_f16_nhwckrsc_nhwc_tilesize128x64x... | 16.0173s | 211.582 µs | GPU 0 | Stream 1165 |
| 17935 | sm80_xmma_fprop_implicit_gemm_f16f16_f16f16_f16_nhwckrsc_nhwc_tilesize128x128... | 16.0197s | 208.958 µs | GPU 0 | Stream 1165 |
| 13331 | mha_v2_1 | 15.1478s | 205.662 µs | GPU 0 | Stream 1165 |
| 18876 | mha_v2_1 | 16.061s | 205.630 µs | GPU 0 | Stream 1165 |
| 13371 | mha_v2_1 | 15.1489s | 205.373 µs | GPU 0 | Stream 1165 |
| 35211 | mha_v2_1 | 18.1491s | 205.341 µs | GPU 0 | Stream 1165 |
| 35676 | mha_v2_1 | 18.1699s | 204.542 µs | GPU 0 | Stream 1165 |
| 18791 | mha_v2_1 | 16.0588s | 204.350 µs | GPU 0 | Stream 1165 |
| 12531 | mha_v2_1 | 15.1095s | 204.286 µs | GPU 0 | Stream 1165 |
| 35171 | mha_v2_1 | 18.148s | 204.254 µs | GPU 0 | Stream 1165 |
| 2076 | mha_v2_1 | 12.0479s | 204.094 µs | GPU 0 | Stream 1165 |
| 12576 | mha_v2_1 | 15.1106s | 204.094 µs | GPU 0 | Stream 1165 |
| 38151 | mha_v2_1 | 18.2863s | 204.029 µs | GPU 0 | Stream 1165 |

# STEP 9

## Use TensorRT 9.1

- Can we further accelerate the pipeline.
- Use Tensorrt 9.1 to inference the model.
- Inference time is 393ms -> 390 ms.



# STEP 10

Move the scheduler logic out.

- Fuse the scheduler logic can improve the performance, but we need to rebuild engine whenever we need to change the scheduler.
- So, we move the scheduler out and following method can help to accelerate the scheduler:
  - Precompute all the parameters which can be compute before reconstruction.
  - Move all the parameters on cuda before reconstruction.
  - Try torch script(https://pytorch.org/docs/stable/generated/torch.jit.script.html )
- The inference speed is 390ms -> 392ms now.

# REFERENCES

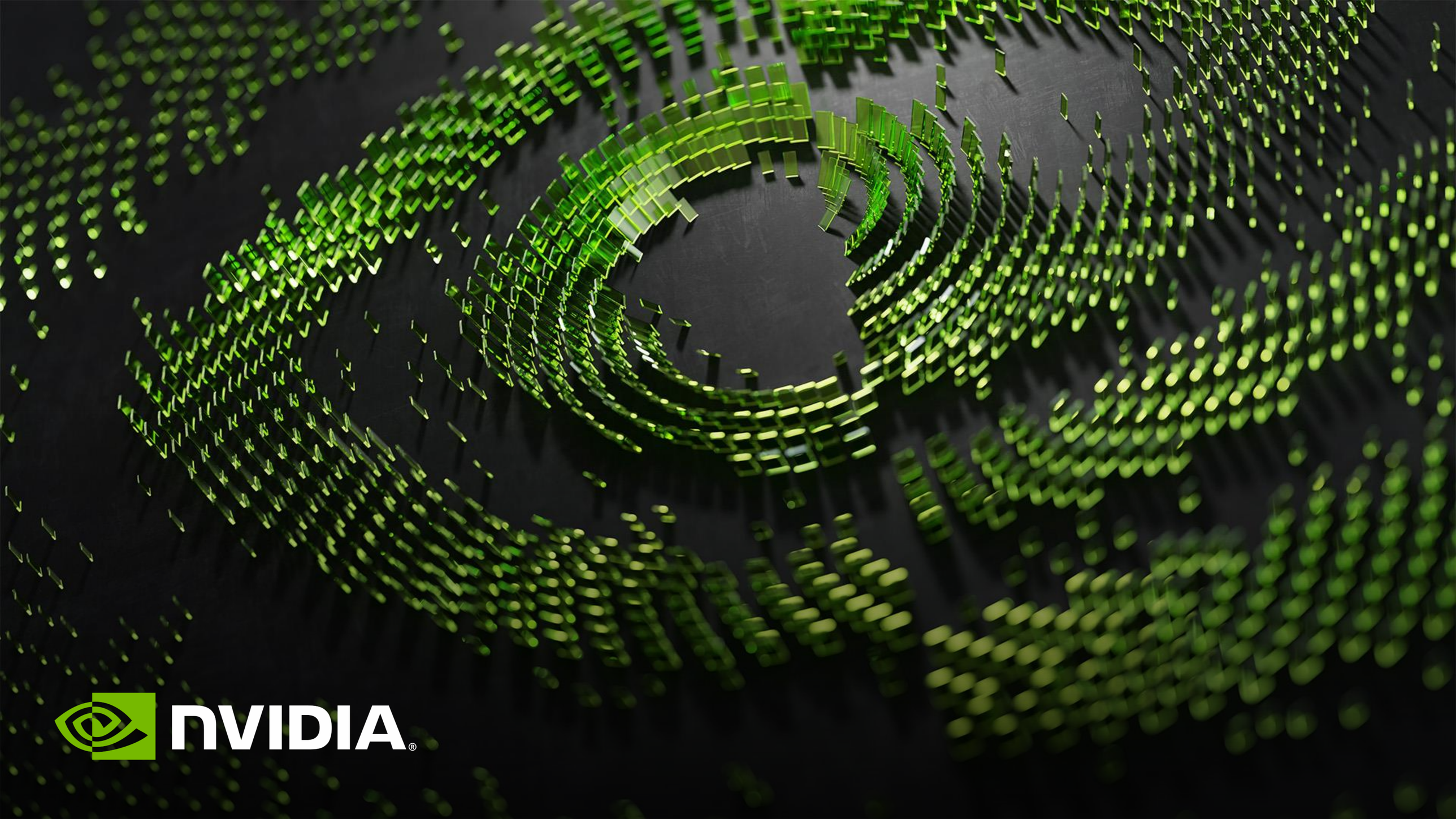

https://github.com/NVIDIA/trt-samples-for-hackathon-cn/tree/master/Hackathon2023/controlnet

https://gitee.com/xingwg/controlnet

https://github.com/xiatwhu/trt2023/tree/master

https://tianchi.aliyun.com/forum/post/569424

https://tianchi.aliyun.com/forum/post/574634


NVIDIA