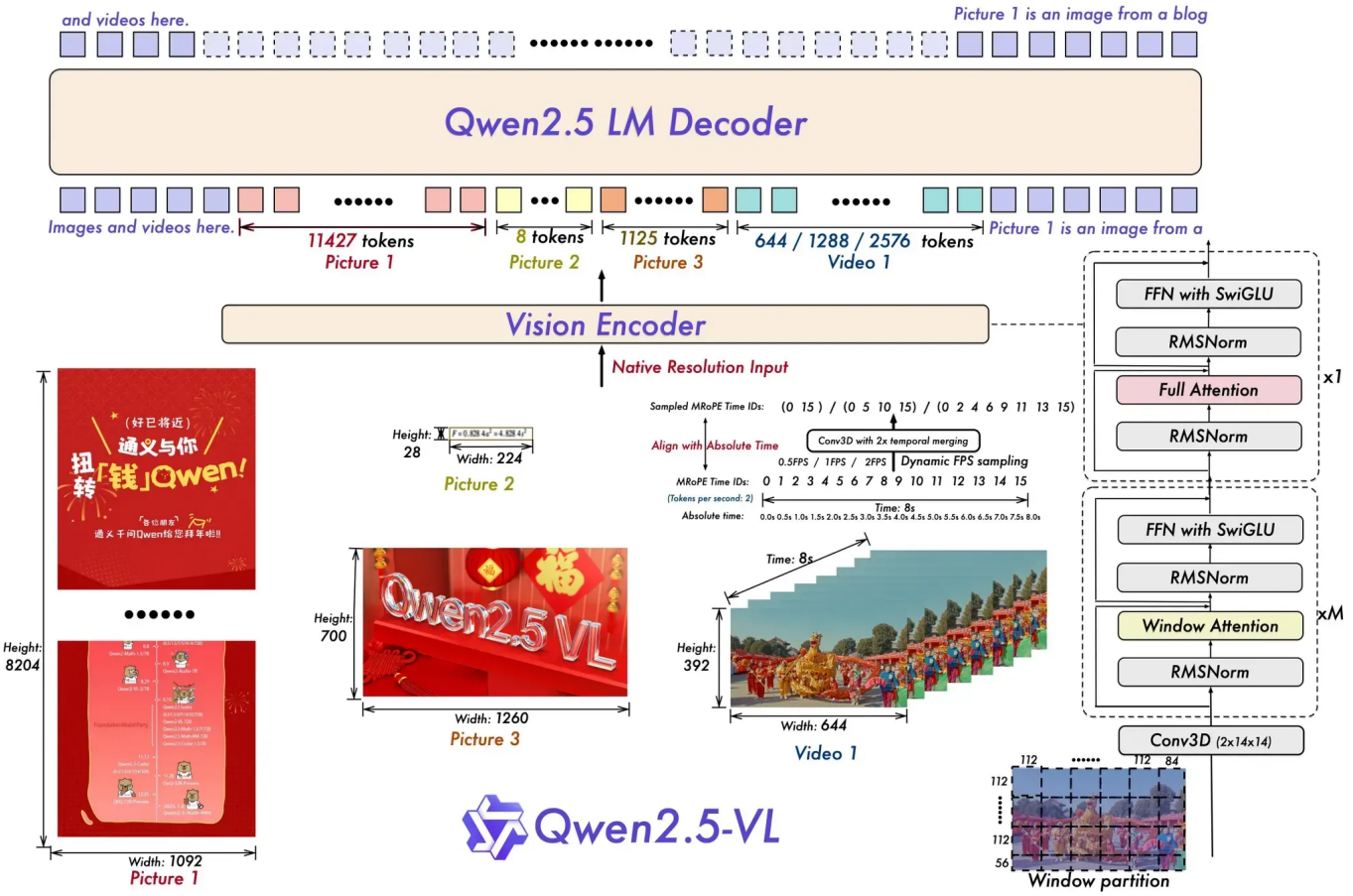
2025.11.27组会

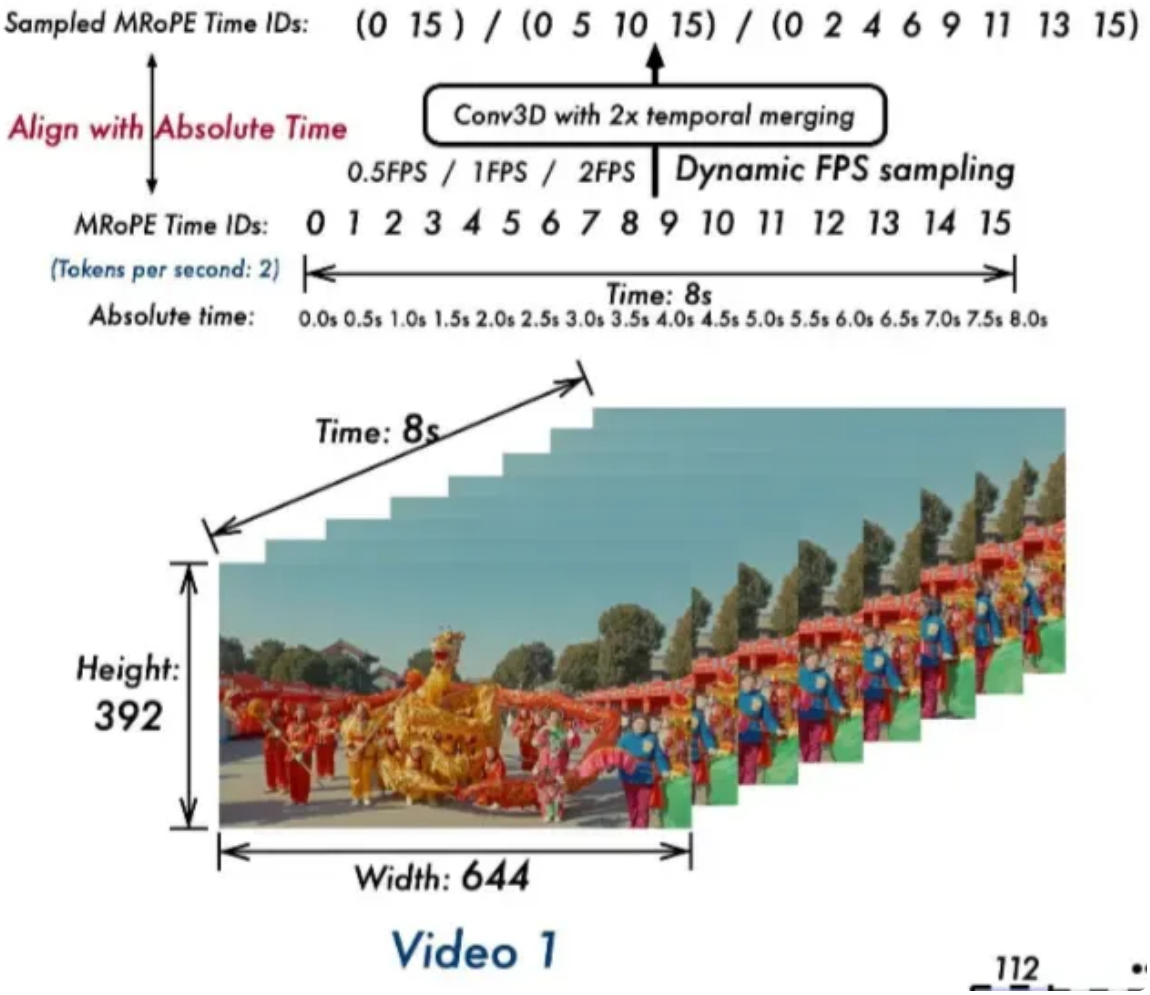


视频 数据流

1. process_vision_info预处理部分
    a. 功能：**输入**的是视频路径，**输出**的是video_inputs包含视频像素数据的列表，`video_inputs[0].shape=[16,3,1008,560] sample_fps=[1.927]`，，分别是采样得帧数、通道数、高、宽。
    b. 阶段流程
        i. 读取视频raw数据(720p 30fps)，`video.shape=[249,3,1280,720]` `info={'video_fps': 30.0}`
        ii. smart_nframes，代码默认`FPS=2`，计算得采样后总帧数`nframes=16`
        iii. 按照采样得总帧数在raw视频的249帧中平均抽取nframes帧得到`video.shape=[16,3,1280,720] sample_fps=1.927`，图例中Sampled MRoPE Time IDs由`idx = tor`

`ch.linspace(0, total_frames - 1, nframes).round().long()` 计算得来

iv. 对采样得到视频的每一帧（图片）进行类似于图片数据流的*“smart_resize”*操作最终得到 `video_inputs.shape=[16,3,1008,560]`



c. 注

i. 上面的 **FPS 和 sample_fps 要进行区分**。FPS 按字面意思**粗略**理解为每秒要采的帧数，但最终采完得到的每秒帧数为 sample_fps。如何来对应呢?

1. FPS的主要初衷是想通过 （ `视频原始总帧数/视频原始帧率 x FPS 对2的倍数取整` ， `249/30 x 2 对2的倍数取整` ）来得到nframes采样后总帧数。
2. 然后在所有原始帧里（0-248）进行采样，抽取nframes帧。此时不一定是一秒2帧了，所以要重新计算 `sample_fps = (video_fps/total_frame)*nframes = (30/249)*16`
3. 这样的结果能够**适配不同帧率**的视频，FPS 可以当作配置参数自己调整以完成**动态帧采样**

2. Qwen2_5_VL_Processor部分

a. 输入： `video_inputs.shape=[16,3,1008,560] sample_fps=[1.927]` 输出： `pixel_values_videos.shape=[23040, 1176]` `video_grid_thw=[[8,72,40]]` `se`

`cond_per_grid_ts=[1.0375]`

i. `second_per_grid_ts **=** (1**/**sample_fps) ***** temporal_patch_size` 即相邻两个时间维度patch之间的时间间隔

b. 实现过程和源码也是对时间、高、宽三个维度分patch

image 数据流

1. image 先经过预处理(process_vision_info) **将长宽近似成 28 的倍数，并resize到不大不小**（[<PIL.Image.Image image mode=RGB size=392x224 at 0x7FEE8DB51660>]，高 `224` ，宽（长） `392` ）

> **process_vision_info对每个image都做了如下处理。** （1）检查每张图片的 `max(h,w) / min(h,w)` 是否在阈值范围内，如果超过阈值。则认为该图片高宽比太离谱，会直接抛出异常（当前阈值200）
>
> （2）通过近似的方式，重新设置图片的 h 和 w 值，确保它们可以被28整除
>
> （3）如果这张图片太大**(分辨率，长x宽），**超过了上述 max_pixels，min_pixels 的范围，那么就在**尽量维持其宽高比例不变的情况下，重新计算其符合max_pixels范围的h和w。图片太小也是同理。** （4）经过前面的步骤，我们得到了这张图片最终理想的h和w值（resized_height，resized weight），我们采用
>
> resize的方式把图片按这个值缩放，就得到image_inputs中的每一个图片
>
> **你可以发现：**
>
> - 这里没有经过任何的“多裁少pad到固定分辨率的操作”，你只是在合理的范围内缩放图片，尽量保存了原始图片的信息。你允许各图片的分辨率各不相同。
> - 每一张图片所含的patch数量是不同的（也就是要送入vit的输入tokens数量是不同的）。
> - **以上两者共同表达了qwenvl中动态分辨率的含义**

2. 将上面的图像([<PIL.Image.Image image mode=RGB size=392x224 at 0x7FEE8DB51660>])经过 `Qwen2VLImageProcessor` ，得到 `pixel_values.shape=[448,1176]` ， `image_grid_thw=[[1,16,28]]` ，数据验证 `448==16*28，16==224/14，28=392/14` 。相当于将每个方格内 `(14x14)` 的像素变成一个 `1176` 维度的向量 `（1176==14*14*3*2）` .
Qwen2VLImageProcessor继承自BaseImageProcessor，通过__call__调用preprocess方法 **(1) do_resize / do_rescale / do_normalize** ： 根据配置决定是否要做这3个操作，分别表示调整图片大小 / 将像素值缩放到0-1之间（乘上1/255）/ 在每个channel上指定mean和std做normalize。 **这**

边的do_resize其实应该在上面process_vision_info中就做过了，也就是这里的操作和之前是有点重复的。（2）把每张图片复制temporal_patch_size次（默认为2）。这是为了在image数据上也增加 T 这个维度，保证image和video的处理逻辑一致（因为video也是把相邻的2帧组成一组）**（3）切分patch，这里patch不是按照一张图从左到右，从上到下的顺序排列的，而是按照把2 * 2 区域内的4个patch变成连续的4个patch排列的。**（可验证）**图像缩放（rescale）**

关键参数 rescale_factor=1/255：图像像素值一般是 0~255 的整数（比如 RGB 图像）。scale=1/255 会让像素值被缩放为 0.0~1.0 的浮点数（等价于 像素值除以 255 ）。这样做的目的是让数值更贴近模型训练要求（多数深度学习模型喜欢较小的输入范围，比如 [0,1] 或 [−1,1] ）。

图像归一化（normalize）

`归一化后像素值 = （原像素值 − 均值 mean）/ 标准差 std`

```
"image_mean": [
      0.48145466,
      0.4578275,
      0.40821073
    ],
   "image_std": [
      0.26862954,
      0.26130258,
      0.27577711
    ],
```

这样可以让图像数据分布更稳定（比如让均值接近 `0` 、标准差接近 `1` ），帮助模型更快收敛、避免梯度异常。

`image_mean` 和 `image_std` （图像均值和标准差）是通过对**训练数据集**的所有图像像素值进行统计计算得到的，目的是让预处理符合数据的真实分布特性。

图片的话需要整体复制一份，补上时间维度，和视频统一格式，在时间维度上合并

- **关键参数** `rescale_factor=1/255` ：图像像素值一般是 `0~255` 的整数（比如 RGB 图像）。`scale=1/255` 会让像素值被缩放为 `0.0~1.0` 的浮点数（等价于 **像素值除以 255** ）。这样做的目的是让数值更贴近模型训练要求（多数深度学习模型喜欢较小的输入范围，比如 `[0,1]` 或 `[−1,1]` ）。

3. 将 `pixel_values` 和 `image_grid_thw` 送给 `VisionEncoder (VisionTransformer)` ，得到输出 `image_embeds.shape==[112, 3584]`
   a. 分patch做embed（3D卷积），维度由 `[448,1176]` 变化为 `[448, 1280]`

图片加载失败

i. 使用3D Conv将原始图像patch转变为vit的输入token

ii. 将hidden_states变为原来的维度形状 `[448,3,2,14,14]`，448是patch个数，3通道数，2时间维度，14*14是patch的大小

iii. `Conv3d(in_channels=3, embed_dim=1280, kernel_size=(2, 14, 14), stride=(2, 14, 14), bias=False)`

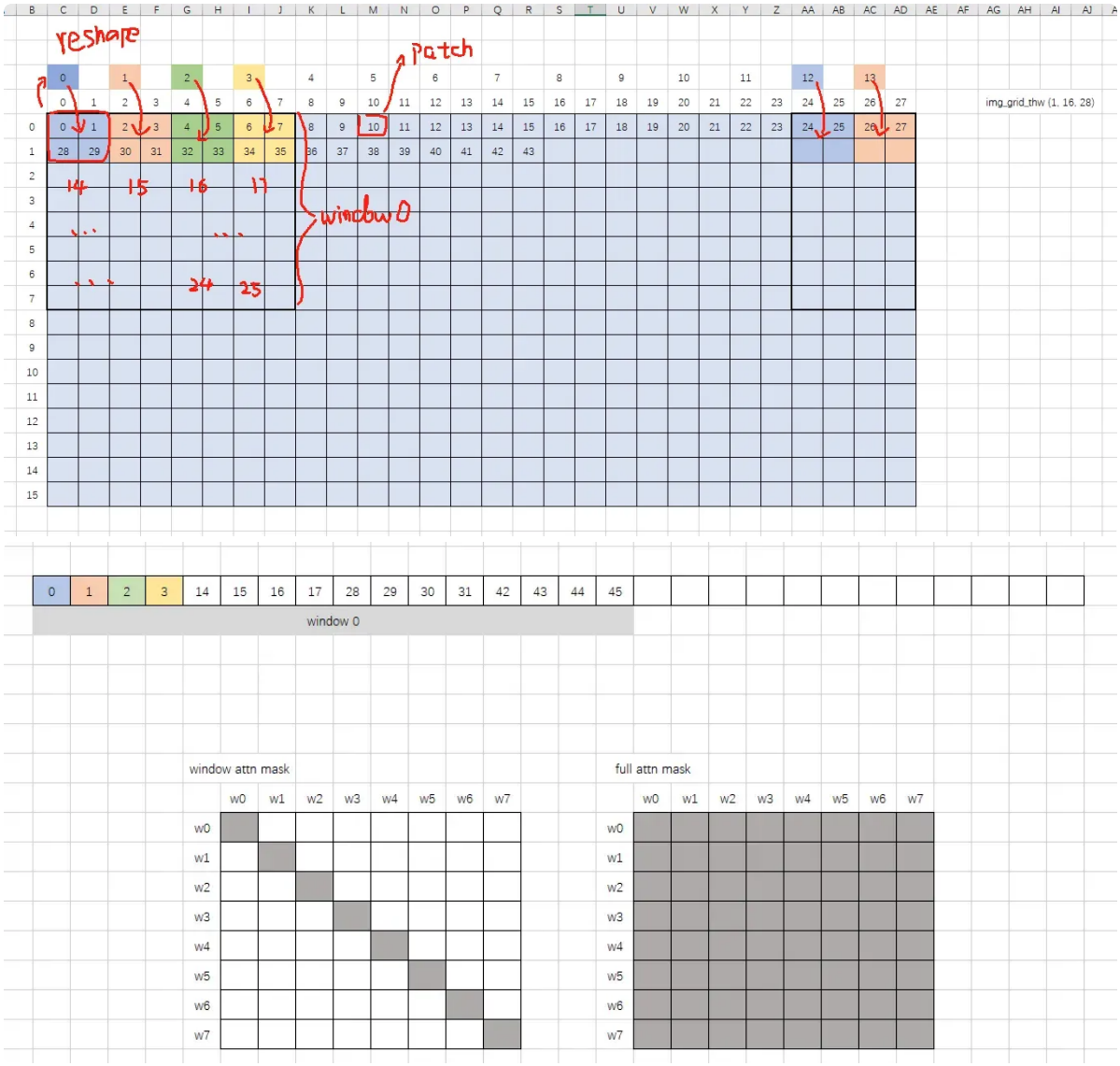
iv. 3D conv要做的事情就是（相当于拿out_channels（= vit_hidden_size）个3D kernel，每个kernel都在所有patch方块上滚一遍， 得到hidden_size的一个数值，所有out_channels个kernel滚完就得到了完整的hidden_size）

```python
class Qwen2_5_VisionPatchEmbed(nn.Module):
    def __init__(
        in_channels: int = 3,
        embed_dim: int = 1152,
    ) -> None:
        super().__init__()
        self.patch_size = patch_size
        self.temporal_patch_size = temporal_patch_size
        self.in_channels = in_channels
        self.embed_dim = embed_dim

        kernel_size = [temporal_patch_size, patch_size, patch_size]
        self.proj = nn.Conv3d(in_channels, embed_dim, kernel_size=kernel_size, stride=kernel_size, bias=False
        # Conv3d(3, 1280, kernel_size=(2, 14, 14), stride=(2, 14, 14), bias=False)

    def forward(self, hidden_states: torch.Tensor) -> torch.Tensor:
        target_dtype = self.proj.weight.dtype
        hidden_states = hidden_states.view(
            -1, self.in_channels, self.temporal_patch_size, self.patch_size, self.patch_size
        )
        hidden_states = self.proj(hidden_states.to(dtype=target_dtype)).view(-1, self.embed_dim)
        return hidden_states
```

b. 经过ViT的VisionBlock各层，维度不变 `[448, 1280]`

i. 在window attention（窗口注意力）前会将hidden_states进行重排序

```
Qwen2_5_VisionTransformerPretrainedModel(Qwen2_5_VLPreTrainedModel):
f forward(self, hidden_states: torch.Tensor, grid_thw: torch.Tensor) -> torch.Tensor:
        device=hidden_states.device,
        dtype=grid_thw.dtype if torch.jit.is_tracing() else torch.int32,
    )
    cu_window_seqlens = torch.unique_consecutive(cu_window_seqlens)

    seq_len, _ = hidden_states.size()
    hidden_states = hidden_states.reshape(seq_len // self.spatial_merge_unit, self.spatial_merge_unit, -1)
    hidden_states = hidden_states[window_index, :, :]
    hidden_states = hidden_states.reshape(seq_len, -1)
    rotary_pos_emb = rotary_pos_emb.reshape(seq_len // self.spatial_merge_unit, self.spatial_merge_unit, -1)
    rotary_pos_emb = rotary_pos_emb[window_index, :, :]
    rotary_pos_emb = rotary_pos_emb.reshape(seq_len, -1)
    emb = torch.cat((rotary_pos_emb, rotary_pos_emb), dim=-1)
    position_embeddings = (emb.cos(), emb.sin())
```



c. 最后经过merger层，维度变化为 `[112, 3584]` （相邻的两个patch融合， `112=448/2/2` ）

   i. 在整个vit的处理中，我们都是使用14*14的patch作为一个token处理的，但是最终进入到LLM中时，我们需要将原来2*2个patch合并成一个token，这可以通过先把2*2个token的结果concat起来，然后经过一个mlp层做hidden_size维度的映射得到

```python
class Qwen2_5_VLPatchMerger(nn.Module):
    def __init__(self, dim: int, context_dim: int, spatial_merge_size: int = 2) -> None:
        super().__init__()
        self.hidden_size = context_dim * (spatial_merge_size**2)
        self.ln_q = Qwen2RMSNorm(context_dim, eps=1e-6)
        self.mlp = nn.Sequential(
            nn.Linear(self.hidden_size, self.hidden_size),
            nn.GELU(),
            nn.Linear(self.hidden_size, dim),
        )

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        x = self.mlp(self.ln_q(x).view(-1, self.hidden_size))
        return x
```