系统分析与设计报告（第7-8周 初版）

目录

项目背景与目标

目标：在现有 Diffusers 式项目结构上，设计并实现若干扩展，使其支持控制视频输入（含相机外参注入）与一种用于加速采样的分段线性流调度器（Perflow）。

约束：保持 `DiffusionPipeline` 与 `SchedulerMixin` 的接口兼容；优先可用性与易调试；将复杂功能以可选组件形式实现，避免破坏现有 pipeline 行为。

成功衡量：接口兼容；模块化实现且可独立测试；提供边界测试与性能对比。

模块级分析

根据 `slide.typ` 中给出的结构，关键模块如下：

- Models：UNet、VAE、Transformer 等，继承自 `ModelMixin` / `ConfigMixin`。核心职责：从噪声/潜在表示预测去噪输出或特征。
- Schedulers：实现噪声时间步及采样更新规则（`SchedulerMixin`），例如 `DDPMScheduler`、`DDIMScheduler`。新增：`PerflowScheduler`。
- Pipelines：继承自 `DiffusionPipeline`，负责端到端推理（文本编码、控制信号注入、去噪循环、解码、后处理）。新增：`WanFunControlPipeline`。
- Loaders/Config：负责从磁盘/仓库加载模型权重与调度器状态（支持 `from_pretrained(..., subfolder="scheduler")`）。
- Utilities：图像/视频 I/O、control encoder、patchify、相机外参转换、metrics、logger。

数据流（高层）：
prompt/text -> tokenizer -> text_encoder -> conditioning
control_video/image -> preprocess -> control_encoder/vae.encode -> control_latents
text_condition + control_latents -> unet 去噪循环 (由 scheduler 控制步进)

-> vae.decode -> frames -> video_bytes

核心流程设计

这里给出两个关键流程的序列式说明：Pipeline 推理流程 与 Scheduler 的 step 更新流程。

1) 推理（WanFunControlPipeline）

Contract（输入/输出简述）

- 输入：
  - prompt: str
  - control_video: Tensor(B, T, C, H, W) 或 视频路径
  - camera_params: list 或 Tensor (长度为 T 或 1)
  - mask: Tensor 可选
  - num_inference_steps, guidance_scale, device 等
- 输出：
  - frames: list[Tensor] 或 video_bytes
  - meta: dict (包含 control_strategy、steps、device、timings)

主要步骤：

1. check_inputs(): 校验控制视频帧数、分辨率、相机外参长度一致性与类型；不满足抛 ValueError。
2. preprocess_control_video(): 解码/resize/normalize/转换.dtype/转 device；输出 Tensor(B,T,C,H,W)。
3. prepare_control_latents(): 调用 control encoder 或 vae.encode，把 control 输入映射到潜在空间；支持 mask；可能抛出 OOM。
4. patchify() + camera 注入：将 control latents 通过 patch_embedding 映射；若 control_adapter 存在，则将 camera params 先通过 control_adapter 映射，然后以加/拼接方式注入 patch embedding；形状不匹配抛 RuntimeError("Shape Mismatch")。
5. 去噪循环：在每个 timestep，调用 unet(model_input, timestep, encoder_hidden_states=cond)，在 model_output 中注入控制信号（可配置注入策略：concat/add/cross-attn）。若未定义策略，使用 fallback 策略并记录 warning。
6. 解码 & 后处理：vae.decode(next_latents) -> frames -> video_bytes。保存或返回 frames。
7. meta 收集：记录 control_strategy、采样步数、耗时、设备信息。

错误与保护：

- 内存不足：捕获 torch.cuda.OutOfMemoryError，尝试释放缓存并降批或降精度（fallback）。
- 输入不一致：ValueError 提示具体字段。
- 数值不稳/shape mismatch：抛 RuntimeError 并记录上下文。

性能注意：

- 支持半精度（float16）与混合精度以减少显存。

- 控制模块可选加载（lazy load），默认关闭以节省资源。

2) Scheduler 更新（PerflowScheduler）

Contract（核心函数）

- set_timesteps(num_inference_steps): 计算内部 timesteps 列表（支持不同采样长度），并构造分段区间 K。
- step(model_output, timestep, latents): 返回 next_latents
- state_dict() / load_state_dict(): 保存/加载 learned theta 和 config

思路：把 time domain 拆分为 K 段，在每一段内部用线性流近似（learned theta）对模型输出做修正，以减少需要的步骤。

数值稳健性保护：

- 当 Perflow 修正导致数值不稳定或相较 baseline（如 DDIM）差异过大时，回退到 baseline 更新。
- load_state_dict 支持 strict=False 的部分加载，缺失 learned theta 时使用零初始化并记录 warning。

高级设计意图

设计原则（来自 slide.typ 的设计哲学）：

- 可用性优先于性能：默认以 float32 在 CPU 上能加载成功，随后可选转为 GPU/float16。
- 易调试 & 易贡献：避免过度抽象，采用单文件策略以便研究者快速修改。
- 明确错误优于自动修正：出错立即暴露并提示原因（除非明确给出 fallback）。
- 插件式可选组件：控制模块与 Perflow 调度器均以可选组件方式实现，保持 backward compatible。

接口设计要点：

- 保持向后兼容：所有新 pipeline 支持 `from_pretrained`, `save_pretrained`, `__call__`, `to`。
- 轻量可选依赖：避免强制引入 heavy runtime，使用 runtime 检测 & optional imports。

安全与资源策略：

- 提供 OOM 处理路径（降精度/分帧处理/提示）。
- 日志级别与诊断信息应足够（timings、tensor shapes、memory usage）。

扩展点 1：WanFunControlPipeline（功能扩展）

目标：实现支持控制视频/图像输入及相机外参注入的扩散 Pipeline。

需求清单（功能性）

- 支持输入类型：Tensor(B,T,C,H,W)、视频路径（自动 decode）、单帧 image。
- 预处理：resize、normalize、帧对齐（若输入帧数与期望步骤数不一致，支持复制/插帧/采样策略）。
- 支持 mask：在 control_latents 上 mask 掩盖与替换。

- 注入策略：支持 "add", "concat", "cross_attn"，可由 pipeline config 指定。
- 相机外参注入：支持 per-frame 或 global 外参，使用 control_adapter 映射到与 patch embedding 一致的形状并逐元素融合。
- 错误检测：输入维度、帧数、外参长度不一致时抛 ValueError；若 shape mismatch（注入后）抛 RuntimeError。

需求清单（非功能性）

- 可配置精度（float32/float16）与设备切换。
- 可选 lazy load 控制模块，减少初始内存占用。
- 支持批量推理（batching）与分帧流水线（streaming）以降低峰值内存需求。

接口与数据形状（示例）

- preprocess_control_video(path_or_tensor) -> Tensor(B, T, C, H, W, dtype=float32)
- prepare_control_latents(control_tensor, mask=None) -> Tensor(B, L, D) # L 为 patch 数，D 为 embedding dim
- patchify(x: Tensor(B,L,D), camera_latents: Optional[Tensor(B,L,D)]) -> Tensor(B,L,D)

注入示例伪码：

```
def patchify(self, x: torch.Tensor, control_camera_latents_input: torch.Tensor | None = None):
  x = self.patch_embedding(x)
  if self.control_adapter is not None and control_camera_latents_input is not None:
    y_camera = self.control_adapter(control_camera_latents_input)
    x = [u + v for u, v in zip(x, y_camera)]
  return x
```

（此为 `slide.typ` 中引用的示例，沿用其简洁策略）

验收测试（最小集）

- 输入单帧 image 路径 -> 正常返回 frames
- 输入 video Tensor(B,T,C,H,W) 与 camera_params 长度不匹配 -> 抛 ValueError
- 注入策略配置为未知字符串 -> 使用 fallback 并记录 warning

扩展点 2：PerflowScheduler（非功能优化 / 加速器）

目标：在兼容 `SchedulerMixin` 签名的前提下，实现分段线性流加速策略，降低采样步数或提高同步质量。

需求要点：

- **init**(config)：包含 K（段数）、每段内部细分、theta 初值与学习率等超参。
- set_timesteps(num_inference_steps)：根据 num_inference_steps 建立 timesteps 列表与分段映射。
- step(model_output, timestep, latents)：基于 Perflow 近似修正预测，返回 next_latents；若数值异常则回退 baseline。
- state_dict() / load_state_dict()：存取 learned theta 与 config，支持 strict=False 部分加载。

- from_pretrained(..., subfolder="scheduler") 支持读取 scheduler/config.json 与权重文件。

数值稳定性保障：

- 在 step() 中监测修正幅度（例如 L2 距离或能量指标），若超过阈值则回退到 baseline（如 DDIM step）。
- 在训练或微调时限制 theta 更新幅度（clip）以避免发散。

测试要点：

- set_timesteps() 边界测试（num_inference_steps 为 1, 2, 非常大值）。
- state_dict roundtrip 测试：保存并加载后，theta 与 config 一致或合理回退。
- 与 baseline 的质量/速度对比：在小规模数据集上统计 PSNR/LPIPS 与采样时间。

测试、验收标准与风险

测试矩阵（最小覆盖）

- 单元测试：
  - `check_inputs()` 边界与异常（shape、帧数不匹配）。
  - `preprocess_control_video()` 对不同输入类型的结果一致性。
  - `patchify()` 在有/无 control_adapter 的路径上的行为。
  - `PerflowScheduler.set_timesteps()` 与 `step()` 输出数值边界。
- 集成测试：在小型模型（e.g., MNIST/小分辨率合成视频）上跑 end-to-end 推理，检查输出 frames 形状与 meta。
- 性能测试：比较 baseline 调度器（DDIM/ EulerDiscrete）与 Perflow 在相同步数与不同步数下的时间与质量。

验收标准（示例）

- 功能正确：WanFunControlPipeline 在合法输入下能完成推理并返回 frames 与 meta。
- 接口兼容：保持 `DiffusionPipeline` 常用接口。
- Perflow 无回退时相比 baseline 能在少量步数下保持相似的质量（需在小实验中达成近似阈值）。

风险与应对

- 数值不稳定：引入回退机制与 clip；增加监控。
- 内存不足：提供降精度与分帧策略。
- 接口破坏：新增功能以可选组件形式实现，默认不开启。

实施计划（建议，里程碑）

- 第 0 周（准备）：熟悉代码库、定位插入点（`src/diffusers/pipelines`, `src/diffusers/schedulers`）。

- 第 1 周：实现 `WanFunControlPipeline` 的骨架（接口、check_inputs、preprocess、prepare_control_latents、简单 patchify）。
- 第 2 周：在小型模型上完成 end-to-end 集成测试与 bug 修复；增加注入策略支持。
- 第 3 周：实现 `PerflowScheduler` 初版（set_timesteps、step、state_dict）、并做数值稳定性测试。
- 第 4 周：性能对比实验、文档与示例、收尾与提交。

交付物（初版）

- `report0/system_analysis_design_report_week7-8.md`（本文件）
- `src/diffusers/pipelines/wan_fun_control.py`（Pipeline 实现骨架）
- `src/diffusers/schedulers/scheduling_perflow.py`（PerflowScheduler 初版）
- tests/ 单元与集成测试（覆盖关键路径）

参考资料

- `report0/slide.typ`（项目需求与设计来源）
- Diffusers 官方设计思想（Models / Pipelines / Schedulers）
- 相关示例：`CogVideoXFunControl`（用于 control video 处理逻辑参考）

附录：关键接口草案

1. WanFunControlPipeline.**call** signature

```
def __call__(
    self,
    prompt: str,
    control_video: Union[str, Tensor],
    camera_params: Optional[List[dict]] = None,
    mask: Optional[Tensor] = None,
    num_inference_steps: int = 50,
    guidance_scale: float = 7.5,
    device: Optional[str] = None,
    return_dict: bool = True,
) -> Tuple[List[Tensor], Dict]:
    ...
```

2. PerflowScheduler.step signature

```
def step(self, model_output: torch.Tensor, timestep: int, latents: torch.Tensor) ->
torch.Tensor:
    """返回 next_latents，内部包含 perflow 修正与回退逻辑"""
```

下一步建议

- 按实施计划由浅入深实现 `WanFunControlPipeline` 骨架与 `PerflowScheduler` 初版。
- 编写最小可运行的单元测试与 end-to-end 集成测试以快速验证设计假设。
- 在课堂研讨会上收集反馈并迭代本报告（补充详细 API 示例与数据转换代码）。