# PR #27432 完整报告

sgl-project/sglang

[diffusion] Fix native text-encoder loading for T5/UMT5 encoder-decoder models

合并时间：2026-06-08 12:23

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/27432

## 执行摘要

- 一句话：修复 T5/UMT5 文本编码器回退加载失败
- 推荐动作：值得阅读，展示了如何优雅处理 transformers 中 encoder-decoder 模型与文本编码器的不兼容问题。设计模式（静态方法 + 配置映射）简洁且易扩展。测试覆盖充分，可作为类似兼容性问题的参考。

## 功能与动机

Wan2.1/Wan2.2 使用 UMT5 文本编码器。当 SGLang 原生加载不可用时，回退路径使用 AutoModel 错误加载了完整的 seq2seq 类（T5Model/UMT5Model），其 forward 需要 decoder 输入导致 ValueError。修改后从 config.architectures 中解析编码器专用类。

## 实现拆解

1. 修改导入：移除顶层 from transformers import AutoModel，改为在 _resolve_transformers_text_encoder_class 方法内局部导入。
2. 新增静态方法：TextEncoderLoader._resolve_transformers_text_encoder_class 负责判断配置 is_encoder_decoder，并将已知的完整 seq2seq 架构名映射为编码器专用类（如 T5Model -> T5EncoderModel），否则回退到 AutoModel。
3. 修改 load_native：将直接调用 AutoModel.from_pretrained 改为先调用解析方法获取具体类，再调用该类的 from_pretrained。
4. 配套测试：新增 test_text_encoder_loader.py，通过 mock.patch 模拟配置加载，覆盖 T5、UMT5、MT5 系列编码器专用类解析，以及非 encoder-decoder、未知架构、配置加载失败的回退行为。

关键文件：

- python/sglang/multimodal_gen/runtime/loader/component_loaders/text_encoder_loader.py（模块 扩散模型；类别 source；类型 dependency-wiring；符号 _resolve_transformers_text_encoder_class）：核心源文件，修改了加载逻辑，新增了映射方法 _resolve_transformers_text_encoder_class
- python/sglang/multimodal_gen/test/unit/test_text_encoder_loader.py（模块 文本编码器；类别 test；类型 test-coverage；符号 TestTextEncoderClassResolution, _resolve, test_umt5_encoder_decoder_uses_encoder_only_class, test_t5_encoder_decoder_uses_encoder_only_class）：新增测试文件，验证各种架构映

射和失败场景

关键符号：_resolve_transformers_text_encoder_class, load_native

## 关键源码片段

### python/sglang/multimodal_gen/runtime/loader/component_loaders/text_encoder_loader.py

核心源文件，修改了加载逻辑，新增了映射方法 _resolve_transformers_text_encoder_class

```
# text_encoder_loader.py ( 核心修改 )
import transformers
from transformers import AutoConfig, AutoModel

@staticmethod
def _resolve_transformers_text_encoder_class(component_model_path, server_args):
    """Resolve the concrete transformers class for a text encoder.

    AutoModel maps encoder-decoder model types (T5/UMT5) to full seq2seq classes,
    which require decoder inputs and fail when used purely as a text encoder.
    For such checkpoints, parse the encoder-only class or map from full seq2seq names.
    """
    try:
        config = AutoConfig.from_pretrained(
            component_model_path,
            trust_remote_code=server_args.trust_remote_code,
            revision=server_args.revision,
        )
    except Exception:
        # config 加载失败时回退到 AutoModel ， 与之前行为一致
        return AutoModel

    if getattr(config, "is_encoder_decoder", False):
        # 定义从完整 seq2seq 架构到编码器专用类的映射
        encoder_only_map = {
            "T5Model": "T5EncoderModel",
            "T5ForConditionalGeneration": "T5EncoderModel",
            "UMT5Model": "UMT5EncoderModel",
            "UMT5ForConditionalGeneration": "UMT5EncoderModel",
            "MT5Model": "MT5EncoderModel",
            "MT5ForConditionalGeneration": "MT5EncoderModel",
        }
        # 遍历 config.architectures ， 依次尝试解析为编码器专用类
        for arch in getattr(config, "architectures", None) or []:
            encoder_arch = encoder_only_map.get(arch, arch)
            transformers_model_class = getattr(transformers, encoder_arch, None)
            if isinstance(transformers_model_class, type):
                return transformers_model_class
    # 非 encoder-decoder 或未识别架构时， 回退到 AutoModel
```

```
    return AutoModel

# 在 load_native 中使用
def load_native(self, component_model_path, server_args, transformers_or_diffusers):
    if transformers_or_diffusers != "transformers":
        return super().load_native(component_model_path, server_args, transformers_or_
        diffusers)
    encoder_idx = 1 if component_model_path.rstrip("/").endswith("text_encoder_2") else 0
    encoder_dtype = server_args.pipeline_config.text_encoder_precisions[encoder_idx]
    # 使用解析得到的类来加载
    transformers_model_class = self._resolve_transformers_text_encoder_class(component_
    model_path, server_args)
    return transformers_model_class.from_pretrained(
        component_model_path,
        trust_remote_code=server_args.trust_remote_code,
        revision=server_args.revision,
        torch_dtype=PRECISION_TO_TYPE[encoder_dtype],
    )
```

## python/sglang/multimodal_gen/test/unit/test_text_encoder_loader.py

新增测试文件，验证各种架构映射和失败场景

```
# test_text_encoder_loader.py
import unittest
from types import SimpleNamespace
from unittest import mock
import transformers
from sglang.multimodal_gen.runtime.loader.component_loaders.text_encoder_loader import
TextEncoderLoader

class TestTextEncoderClassResolution(unittest.TestCase):
    """验证 load_native 在 encoder-decoder 模型上不会使用 AutoModel 加载"""

    server_args = SimpleNamespace(trust_remote_code=False, revision=None)

    def _resolve(self, is_encoder_decoder, architectures):
        # 创建一个模拟 config 对象
        config = SimpleNamespace(
            is_encoder_decoder=is_encoder_decoder,
            architectures=architectures,
        )
        # 模拟 AutoConfig.from_pretrained 返回该 config
        with mock.patch.object(
            transformers.AutoConfig, "from_pretrained", return_value=config
        ):
            return TextEncoderLoader._resolve_transformers_text_encoder_class(
                "dummy/path", self.server_args
            )

    def test_umt5_encoder_decoder_uses_encoder_only_class(self):
```

```
        self.assertIs(
            self._resolve(True, ["UMT5EncoderModel"]), transformers.UMT5EncoderModel
        )
        self.assertIs(self._resolve(True, ["UMT5Model"]), transformers.UMT5EncoderModel)
        self.assertIs(
            self._resolve(True, ["UMT5ForConditionalGeneration"]),
            transformers.UMT5EncoderModel,
        )

    def test_t5_encoder_decoder_uses_encoder_only_class(self):
        self.assertIs(
            self._resolve(True, ["T5EncoderModel"]), transformers.T5EncoderModel
        )
        self.assertIs(self._resolve(True, ["T5Model"]), transformers.T5EncoderModel)
        self.assertIs(
            self._resolve(True, ["T5ForConditionalGeneration"]),
            transformers.T5EncoderModel,
        )

    def test_mt5_encoder_decoder_uses_encoder_only_class(self):
        self.assertIs(
            self._resolve(True, ["MT5EncoderModel"]), transformers.MT5EncoderModel
        )
        self.assertIs(self._resolve(True, ["MT5Model"]), transformers.MT5EncoderModel)
        self.assertIs(
            self._resolve(True, ["MT5ForConditionalGeneration"]),
            transformers.MT5EncoderModel,
        )

    def test_non_encoder_decoder_keeps_automodel(self):
        # 非 encoder-decoder 模型（如 CLIP ）应返回 AutoModel
        self.assertIs(self._resolve(False, ["CLIPTextModel"]), transformers.AutoModel)

    def test_unknown_architecture_falls_back_to_automodel(self):
        # 未知架构应回退到 AutoModel
        self.assertIs(self._resolve(True, ["NotARealClass"]), transformers.AutoModel)

    def test_config_load_failure_falls_back_to_automodel(self):
        # config 加载失败时应回退到 AutoModel
        with mock.patch.object(
            transformers.AutoConfig,
            "from_pretrained",
            side_effect=OSError("no config"),
        ):
            cls = TextEncoderLoader._resolve_transformers_text_encoder_class(
                "dummy/path", self.server_args
            )
        self.assertIs(cls, transformers.AutoModel)
```

```
if __name__ == "__main__":
    unittest.main()
```

## 评论区精华

Review 中 gemini-code-assist[bot] 建议增加从完整 seq2seq 架构（如 T5ForConditionalGeneration）到编码器专用类的映射，并更新测试覆盖这些情况。此建议已被采纳到最终实现中。另外 mickqian 对变量命名 model_class 建议改为 transformers_model_class，也已被采纳。

- 映射完整 seq2seq 架构到编码器专用类 (design): 已采纳，在最终代码中添加了 encoder_only_map 字典
- 变量命名建议 (style): 已采纳，在最终提交中已修改

## 风险与影响

- 风险：风险较低。主要影响回退加载路径，若 config 加载失败或架构不在映射表中，将回退到 AutoModel，与之前行为一致。测试覆盖了已知的 encoder-decoder 架构和失败路径。但需注意若未来出现新的 encoder-decoder 架构但未在映射表中，可能仍会失败，但会回退到 AutoModel（可能再次遇到相同问题），需持续更新映射表。
- 影响：影响范围限于 diffusion 模块中 T5/UMT5 文本编码器的回退加载路径。其他编码器（CLIP、Mistral、Qwen）完全不受影响。用户使用 Wan2.1/Wan2.2 等模型时，回退路径将正常工作，避免运行时崩溃。
- 风险标记：回退路径变更

## 关联脉络

- 暂无明显关联 PR