

PR #32798 完整报告

sgl-project/sglang

DFLASH support added for XPU

合并时间: 2026-09-11 10:39

原文链接: <http://prhub.com.cn/sgl-project/sglang/pull/32798>

执行摘要

- 一句话: XPU 平台新增 DFLASH 投机解码支持与端到端测试
- 推荐动作: 值得精读, 但重点不在代码量 (源码仅 +36/-19), 而在三处设计决策: 一是 `xpu_backend.py` 把 non-causal 语义从 `is_cross_attention` 扩展到 `ENCODER_ONLY / DECODER_BIDIRECTIONAL`, 这是 DFLASH draft 双向注意力能在 XPU 上工作的前提, 也是本 PR 唯一改动通用注意力路径的地方, 建议与 `FlashAttentionBackend` 逐行对照确认无遗漏分支; 二是 `dflash_utils.py` 中 `_DFLASH_VERIFY_SKIP_CUSTOM_MASK_BACKENDS` 的语义 (只在 `decode graph capture` 时生效) 以及它与 `topk==1` 的关系, 这决定了新增的 `decode-graph` 用例是否真能覆盖到新代码; 三是 `draft attention backend` 的默认值选择尚未闭环, 跟进 `mingfeima` 的结论可避免 XPU 用户长期停留在次优后端。测试文件的基类复用与 `sys.path` 处理技巧也值得借鉴。

功能与动机

PR body 明确写着「This PR add support for XPU with DFLASH algo in speculative decoding」。变更前 DFLASH 在参数解析阶段就被硬性挡住:

`speculative_hook.py::_handle_dflash` 只接受 `cuda*/npu`, `draft backend` 候选集合里没有任何 Intel 后端, `XPUAttentionBackend` 的类注释中「Speculative Decoding support」还挂在 TODO 上。因此动机是补齐平台支持而非新增算法: 让已有的 DFLASH (block 级 verify 窗口 + 从 target hidden states 物化 draft KV) 在 Intel GPU 上能起、能跑、有回归用例。讨论还透露两条业务诉求: Valentine233 关心「intel_xpu 的加速比更好, 为什么默认还是 triton」并要求补齐 CUDA 侧 SD 速度数据; mingfeima 关心 XPU 是否支持 `page_size 256`。

实现拆解

1. 放宽设备门禁 (`python/sglang/srt/arg_groups/speculative_hook.py::_handle_dflash`): 条件由 `cuda/npu` 扩展为 `cuda/npu/xpu`, 报错文案同步改为 `CUDA, NPU and XPU`; `DP attention`、`pp_size`、必须指定 `draft` 模型路径等限制保持不变, XPU 走的是和其他平台完全相同的校验链。
2. 收口 `draft attention backend` 解析 (同文件 `_resolve_dflash_draft_attention_backend` + `python/sglang/srt/arg_groups/choices.py` + `python/sglang/srt/speculative/draft_worker_common.py::_resolve_draft_attention_backend_fallback`): 删除本地硬编码的 `supported_draft_backends` 元组, 改为引用 `DRAFT_ATTENTION_BACKEND_CHOICES` 常量, 并在该常量中新增 `intel_xpu`; 两处

fallback 从「仅看 torch.version.hip」升级为 is_xpu 或 is_hip 时回退 triton，否则 flashinfer，原因是 FlashInfer 只在 CUDA 可用。

3. 让 XPU attention 支持 DFLASH 的双向 draft 层（python/sglang/srt/layers/attention/xpu_backend.py）：引入 AttentionType，把 forward_extend 与 forward_decode 中的 causal = not layer.is_cross_attention 改为同时排除 AttentionType.ENCODER_ONLY 与 AttentionType.DECODER_BIDIRECTIONAL，与 FlashAttentionBackend 语义对齐；配套把 XPUAttentionBackend 加入 python/sglang/srt/speculative/dflash_utils.py 的 _DFLASH_VERIFY_SKIP_CUSTOM_MASK_BACKENDS。
4. 启用 XPU 上的 GPU triton / fused-KV 快路径（python/sglang/srt/speculative/dflash_worker_v2.py）：引入 is_xpu，_use_fused_kv_materialize 与 supports_gpu_triton 均补上 is_xpu()，从而带动 _use_triton_prepare_block、_use_triton_accept_bonus、_use_triton_compact_rebuild 三个开关，避免 XPU 继续走每步两次 host-sync 的 legacy compact-rebuild 路径。
5. 测试与 CI 配套（test/registered/e2e/xpu/test_dflash_intel_xpu.py，+156 行）：复用 CUDA 侧 registered.spec.dflash.test_dflash 的测试基类，用 _XPU_LAUNCH_ARGS = ["--device", "xpu", "--mem-fraction-static", "0.75"] 覆盖基类默认启动参数，通过 register_xpu_ci(est_time=600, suite="nightly-xpu-1-gpu", nightly=True) 注册到 nightly；12 个用例类覆盖 triton 与 intel_xpu 两套后端，并按 review 意见补上 --cuda-graph-backend-decode full 的 decode graph 变体。

关键文件：

- python/sglang/srt/arg_groups/speculative_hook.py（模块 投机参数；类别 source；类型 dependency-wiring；符号 _handle_dflash, _resolve_dflash_draft_attention_backend）：变更入口：_handle_dflash 放开 xpu 设备门禁，_resolve_dflash_draft_attention_backend 把 draft backend 白名单收敛到 DRAFT_ATTENTION_BACKEND_CHOICES 并将平台回退升级为 is_xpu or is_hip -> triton。
- python/sglang/srt/layers/attention/xpu_backend.py（模块 注意力后端；类别 source；类型 core-logic；符号 XPUAttentionBackend.forward_extend, XPUAttentionBackend.forward_decode）：唯一改动通用 XPU 注意力路径的文件：把 causal 判定扩展为同时排除 ENCODER_ONLY / DECODER_BIDIRECTIONAL，是 DFLASH draft full_attention 层能在 XPU 上跑通的前提。
- test/registered/e2e/xpu/test_dflash_intel_xpu.py（模块 端到端测试；类别 test；类型 test-coverage；符号 TestDFlashIntelXPU, TestDFlashIntelXPUPage256, TestDFlashIntelXPUPagedPrefill, TestDFlashIntelXPUPagedDecodeGraph）：新增 156 行端到端测试，12 个用例类覆盖 triton 与 intel_xpu 两套后端及 page256 / chunked prefill / decode graph / overlap 变体，并注册到 nightly-xpu-1-gpu。
- python/sglang/srt/speculative/draft_worker_common.py（模块 草稿后端选择；类别 source；类型 dependency-wiring；符号 _resolve_draft_attention_backend_fallback）：_resolve_draft_attention_backend_fallback 的平台回退与 speculative_hook 保持同步，改为 is_xpu or torch.version.hip -> triton。

- python/sglang/srt/speculative/dflash_worker_v2.py (模块 草稿执行; 类别 source; 类型 core-logic; 符号 _use_fused_kv_materialize, supports_gpu_triton, _use_triton_prepare_block, _use_triton_accept_bonus) : 把 XPU 纳入 GPU triton / fused-KV 快路径判定, 决定 DFLASH 在 XPU 上是否会走 fused_kv_materialize 与 triton prepare_block / accept_bonus 内核。
- python/sglang/srt/speculative/dflash_utils.py (模块 验证掩码; 类别 source; 类型 core-logic; 符号 _DFLASH_VERIFY_SKIP_CUSTOM_MASK_BACKENDS) : 把 XPUAttentionBackend 加入 _DFLASH_VERIFY_SKIP_CUSTOM_MASK_BACKENDS; review 指出该集合只影响 decode CUDA-graph capture, 由此引出新增 decode-graph 用例。
- python/sglang/srt/arg_groups/choices.py (模块 参数选项; 类别 source; 类型 configuration; 符号 DRAFT_ATTENTION_BACKEND_CHOICES) : DRAFT_ATTENTION_BACKEND_CHOICES 新增 intel_xpu, 使该常量成为 draft backend 白名单的唯一来源。

关键符号: _handle_dflash, _resolve_dflash_draft_attention_backend, _resolve_draft_attention_backend_fallback, XPUAttentionBackend.forward_extend, XPUAttentionBackend.forward_decode

关键源码片段

python/sglang/srt/arg_groups/speculative_hook.py

变更入口: _handle_dflash 放开 xpu 设备门禁, _resolve_dflash_draft_attention_backend 把 draft backend 白名单收敛到 DRAFT_ATTENTION_BACKEND_CHOICES 并将平台回退升级为 is_xpu or is_hip -> triton。

```
# python/sglang/srt/arg_groups/speculative_hook.py
```

```
def _handle_dflash(server_args: ServerArgs) -> None:
```

```
    cfg = resolving_view(server_args)
```

```
    # 设备门禁从 CUDA / NPU 扩展到 XPU: 这是本 PR 的入口改动,
```

```
    # 在此之前 XPU 用户会在参数解析阶段直接收到 ValueError。
```

```
    if not (
```

```
        cfg.device.startswith("cuda") or cfg.device == "npu" or cfg.device == "xpu"
```

```
):
```

```
    raise ValueError(
```

```
        "DFLASH speculative decoding only supports CUDA, NPU and XPU devices."
```

```
)
```

```
    # 其余限制对 XPU 同样生效, 保证只有一条校验链路:
```

```
    # DP attention、pp_size、必须显式指定 draft 模型路径。
```

```
    if resolved_view(server_args).enable_dp_attention:
```

```
        raise ValueError(
```

```
            "Currently DFLASH speculative decoding does not support dp attention."
```

```
)
```

```
    if cfg.pp_size != 1:
```

```

        raise ValueError(
            "Currently DFLASH speculative decoding only supports pp_size == 1."
        )
    if cfg.speculative_draft_model_path is None:
        raise ValueError(
            "DFLASH speculative decoding requires setting --speculative-draft-model-path."
        )

def _resolve_dflash_draft_attention_backend(server_args: ServerArgs) -> None:
    """把 speculative_draft_attention_backend 解析为一个受支持的最终值。"""
    cfg = resolving_view(server_args)

    # 白名单改为复用 arg_groups.choices 里的单一事实来源（本 PR 在其中新增 intel_xpu），
    # 避免这里与 draft_worker_common 各留一份副本、改一处漏一处。
    supported_draft_backends = DRAFT_ATTENTION_BACKEND_CHOICES
    # FlashInfer 只在 CUDA 可用；XPU 与 ROCm 一律回退到 triton。
    # 这一行把 fallback 从「只看 torch.version.hip」升级为「看平台」。
    fallback_backend = (
        "triton" if (get_platform().is_xpu or get_platform().is_hip) else "flashinfer"
    )

    draft_backend = cfg.speculative_draft_attention_backend
    if draft_backend is None:
        # 用户没指定时沿用进程级 prefill backend；仍为空才落到平台 fallback。
        draft_backend, _ = attention_backends_of(resolved_view(server_args))
    if draft_backend is None:
        draft_backend = fallback_backend
    # 后续 trtllm_mha 的逐层 sliding / causal 判定与本 PR 的平台扩展无关，保持原样。

```

python/sglang/srt/layers/attention/xpu_backend.py

唯一改动通用 XPU 注意力路径的文件：把 causal 判定扩展为同时排除 ENCODER_ONLY / DECODER_BIDIRECTIONAL，是 DFLASH draft full_attention 层能在 XPU 上跑通的前提。

```

# python/sglang/srt/layers/attention/xpu_backend.py
from sglang.srt.layers.radix_attention import AttentionType

class XPUAttentionBackend(AttentionBackend):
    def forward_extend(self, q, k, v, layer, forward_batch):
        # （省略：KV cache 写入、forward_metadata 准备等前置逻辑，均未改动）

        # 窗口大小计算；DFLASH draft 的 full_attention 层不走 SWA，会落到 (-1, -1)。
        is_hybrid_swa = (
            layer.sliding_window_size is not None and layer.sliding_window_size > -1
        )
        window_size = (layer.sliding_window_size, 0) if is_hybrid_swa else (-1, -1)

        # 关键修正：对齐 FlashAttentionBackend —— ENCODER_ONLY / DECODER_

```

```

BIDIRECTIONAL
# （也就是 DFLASH draft 的 full_attention 层）同样是非 causal，
# 而不是只有 cross-attention 才非 causal。缺这一步，draft 的双向注意力
# 会被按 causal 计算，结果与 CUDA 平台不一致。
causal = not (
    layer.is_cross_attention
    or layer.attn_type
    in (AttentionType.ENCODER_ONLY, AttentionType.DECODER_BIDIRECTIONAL)
)

def forward_decode(self, q, k, v, layer, forward_batch):
    # （省略：spec_info / cascade attention 判定等前置逻辑）

    # decode、DRAFT_DECODE 与 target verify 共用同一处判定，
    # 保证 draft 与 verify 两个阶段对同一层的 causal 语义保持一致。
    causal = not (
        layer.is_cross_attention
        or layer.attn_type
        in (AttentionType.ENCODER_ONLY, AttentionType.DECODER_BIDIRECTIONAL)
    )

```

test/registered/e2e/xpu/test_dflash_intel_xpu.py

新增 156 行端到端测试，12 个用例类覆盖 triton 与 intel_xpu 两套后端及 page256 / chunked prefill / decode graph / overlap 变体，并注册到 nightly-xpu-1-gpu。

```

# test/registered/e2e/xpu/test_dflash_intel_xpu.py
"""DFLASH speculative decoding on Intel XPU."""

import os
import sys
import unittest

from slang.srt.utils.common import is_xpu

# 把 test/ 根目录放进 sys.path: CI 直接以 `python3 <full_path>` 运行，
# pytest 也只插入文件自身目录，因此要手动补上三级父目录。
_TEST_ROOT = os.path.abspath(os.path.join(os.path.dirname(__file__), "..", "..", ".."))
if _TEST_ROOT not in sys.path:
    sys.path.insert(0, _TEST_ROOT)

# 以模块属性引用基类，而不是直接 import 类名：pytest 按 class __name__ 收集，
# 直接导入基类会让它在这里被二次收集，把 CUDA / flashinfer 的基类配置也带上。
from registered.spec.dflash import test_dflash as _dflash_base

from slang.test.ci.ci_register import register_xpu_ci

register_xpu_ci(est_time=600, suite="nightly-xpu-1-gpu", nightly=True)

# setUpClass 会把这段参数追加在基类启动参数之后：末尾的 --mem-fraction-static

```

```
# 覆盖基类写死的 0.7, --device 选中 Intel GPU; 需要额外参数的变体必须把自身参数追加在后面
# (基类不做合并, 子类的 other_launch_args 是整体替换)。
_XPU_LAUNCH_ARGS = ["--device", "xpu", "--mem-fraction-static", "0.75"]
```

```
@unittest.skipUnless(is_xpu(), "Intel XPU not available")
class TestDFlashIntelXPU(_dflash_base.TestDFlashServerBase):
    """XPU 默认后端 (triton) 下的完整 DFLASH 套件。"""

    max_running_requests = 8
    attention_backend = "triton"
    other_launch_args = _XPU_LAUNCH_ARGS
```

```
@unittest.skipUnless(is_xpu(), "Intel XPU not available")
class TestDFlashIntelXPUChunkedPrefill(_dflash_base.TestDFlashServerChunkedPrefill):
    """triton 后端下的小 chunk (4) chunked prefill 变体。"""

    max_running_requests = 8
    attention_backend = "triton"
    other_launch_args = _XPU_LAUNCH_ARGS + ["--chunked-prefill-size", "4"]
```

```
@unittest.skipUnless(is_xpu(), "Intel XPU not available")
class TestDFlashIntelXPUDecodeGraph(_dflash_base.TestDFlashServerBase):
    """显式打开 decode CUDA-graph (--cuda-graph-backend-decode full) 的 XPU 用例,
    用于覆盖 _DFLASH_VERIFY_SKIP_CUSTOM_MASK_BACKENDS 中新增的
    XPUAttentionBackend。"""

    max_running_requests = 8
    attention_backend = "triton"
    other_launch_args = _XPU_LAUNCH_ARGS + ["--cuda-graph-backend-decode", "full"]
```

```
# --- 原生 XPUAttentionBackend (intel_xpu) 变体 ---
# intel_xpu 不是 XPU 默认后端 (默认是 triton), 必须显式选择;
# 这批用例守护 DFLASH 的 draft / verify 路径经过 XPUAttentionBackend 的行为。
```

```
@unittest.skipUnless(is_xpu(), "Intel XPU not available")
class TestDFlashIntelXPUBackend(_dflash_base.TestDFlashServerBase):
    """原生 XPUAttentionBackend 下的完整 DFLASH 套件。"""

    max_running_requests = 8
    attention_backend = "intel_xpu"
    other_launch_args = _XPU_LAUNCH_ARGS
```

```
@unittest.skipUnless(is_xpu(), "Intel XPU not available")
```

```
class TestDFlashIntelXPUBackendChunkedPrefill(
    _dflash_base.TestDFlashServerChunkedPrefill
):
    """intel_xpu 受 page size 限制（仅支持 64 / 128），chunk 只能取 128 而不是 4。"""

    max_running_requests = 8
    attention_backend = "intel_xpu"
    other_launch_args = _XPU_LAUNCH_ARGS + ["--chunked-prefill-size", "128"]
```

评论区精华

- siju-samuel 对 dflash_utils.py:39 给出了本 PR 最有价值的一条 review：加入 XPUAttentionBackend 只影响 decode CUDA-graph capture，而 XPU 默认禁用 decode graph，且没有任何用例传 --cuda-graph-backend-decode full，所以该路径此前从未被真正执行；同时 TestDFlashIntelXPUBackendNoCudaGraph 与默认配置重复。结论是「The change itself is correct」——DFLASH 强制 speculative_eagle_topk = 1，xpu_backend.py:288 的 custom_mask 读取受 self.topk > 1 保护，因此 XPU 路径不会触达该掩码——但测试应换成 decode-graph 开启的变体。作者随后补了 TestDFlashIntelXPUDecodeGraph。
- siju-samuel 对 chunked prefill 尺寸提出质疑：「why big size chunk128 and not 4 here? Any issues? I guess small chunks are the entire point of that variant.」作者解释 intel_xpu 受 page size 限制，目前只支持 64/128，所以该变体只能用 128。
- mingfeima 追问「do xpu support pagesize 256?」，作者回复 page_size=256 不被 XPU 后端支持、内部会 override 到 128，Page256 用例仅作 sanity check。
- Valentine233 指出性能与默认值不一致：「According to your perf result, intel_xpu get a better speedup. Is there any specific reason to use triton as the default attn backend?」作者表示先从 triton 做起、可以改默认为 intel_xpu；Valentine233 把决策权交给 mingfeima，并提醒「please check CI failures like lint」。该问题在 PR 内没有最终结论。
- 早期 review 还指出测试基类命名过时（TestDFlashServerSpecV2 / TestDFlashServerSpecV2PlanStream 已不存在），作者确认在最新 main 上已重命名为 TestDFlashServerOverlap / TestDFlashServerOverlapPlanStream 并完成修正。
- _DFLASH_VERIFY_SKIP_CUSTOM_MASK_BACKENDS 加入 XPUAttentionBackend 是否真正被执行 (correctness)：作者回复「testcase with xpu graph enabled is added」，最终版新增 TestDFlashIntelXPUDecodeGraph (--cuda-graph-backend-decode full)，测试覆盖缺口被补上。
- intel_xpu 变体的 chunked prefill 为何用 128 而非 4 (testing)：作者解释 intel_xpu 对 page size 有依赖，目前只支持 64 / 128，因此无法取 4；triton 变体仍保留 chunk 4。
- XPU 是否支持 page_size 256 (question)：作者确认 page_size=256 不被 XPU 后端支持，内部会 override 到 128，Page256 用例仅作 sanity check、可选保留。
- DFLASH draft 默认注意力后端：triton 还是 intel_xpu (design)：PR 内未拍板，合并版本仍以 triton 为 XPU 默认 draft 后端，后续需要跟进维护者决策。

- 测试基类命名过时导致用例无法收集 (testing): 测试文件按最新基类名修正, 最终版本继承 TestDFlashServerOverlap 与 TestDFlashServerOverlapPlanStream。
- CI 失败与 lint 清理 (testing): 作者提交了独立的 Linting commit, PR 最终由 mingfeima 合并; 材料中未提供最终 CI 全绿的证据。

风险与影响

- 风险:

1. 默认后端未定论导致的性能损失: `_resolve_dflash_draft_attention_backend` 在用户未显式指定时, 先取进程 `prefill backend`, 再按平台回退到 `triton`; 而作者自己的数据表明 `intel_xpu` 在 `block_size=12` 时 `decode 5.23x`、`e2e 4.66x`, 优于 `triton` 变体。默认值不变意味着大多数用户拿不到最优配置, 且该选型只在评论区被提出、PR 内未拍板。
2. 注意力语义在 XPU 主路径上被放宽: `xpu_backend.py` 的 `causal` 判定改动同时作用于 `forward_extend` 与 `forward_decode`, 任何 `attn_type` 为 `ENCODER_ONLY` / `DECODER_BIDIRECTIONAL` 的层在 XPU 上都会从 `causal` 变为非 `causal`。虽然与 `FlashAttentionBackend` 对齐是正确方向, 但这是 XPU 通用注意力路径的行为变更, 非 DFLASH 的编码器类模型也需要回归验证。
3. XPU 上首次启用 `triton` 快路径: `dflash_worker_v2.py` 把 `_use_fused_kv_materialize` 与 `supports_gpu_triton` 扩展到 `is_xpu()`, 即 XPU 将首次走 `fused_kv_materialize`、`triton prepare_block`、`triton accept_bonus`、`triton compact_rebuild` 等自定义 kernel。若 XPU 侧 `sgl-kernel/triton` 版本对这些算子支持不完整, 会在运行期才暴露, 缺少编译期或启动期校验。
4. `page_size` 静默降级: `TestDFlashIntelXPUPage256` 实际跑的是被内部 `override` 到 `128` 的配置, 用户显式传 `--page-size 256` 时行为与配置不一致, 可能造成排查困难。
5. 覆盖范围与 CI: 测试依赖 `is_xpu()` 的 `skipUnless` 且注册为 `nightly` (`suite="nightly-xpu-1-gpu"`), 主干 PR 测试不会执行; PR body 中 `pr-test`、`pr-test-extra`、`AMD ROCm 10` 三个状态均为失败 (`:x:`), 讨论中也被要求清理 lint, 材料未给出最终全绿证据。
6. 性能对比基线不对等: XPU 侧为 `graph-off` (24GB 显存下 `decode graph` 不可行), CUDA 侧为 `graph-on`, `CUDA-off/XPU-off` 的差距在 `bs=1/4/8` 分别为 `2.28x/3.33x/4.50x`, 跨设备结论需谨慎解读。
 - 影响: 用户侧: Intel XPU (Arc Pro B60 24GB 等) 用户可以在同一套 DFLASH 参数下获得投机解码能力, 实测 `decode` 加速 `5.23x`、`e2e 4.66x` (`block_size=12`、`graph-off`、`mem-frac 0.75`), 部署时需注意显存比例要抬到约 `0.75~0.88` (`0.75` 时 `bs=8` 的 KV 容量不足会跳过该 batch)。系统侧: `XPUAttentionBackend` 的 `causal` 语义与 `FlashAttentionBackend` 对齐, 属于跨平台一致性修复; `draft backend` 解析的两处副本被收敛到 `DRAFT_ATTENTION_BACKEND_CHOICES` 单一事实来源, 后续加平台只需改一处。团队侧: 新增 12 个用例、`est_time=600` 的 `nightly-xpu-1-gpu` 测试, 长期需要 XPU CI 资源与值班维护; 默认 `draft` 后端选型留给 `mingfeima`, 意味着还有一次后续决策与可能的配置调整。
 - 风险标记: 默认 `draft` 后端未定论, 注意力语义变更, XPU 首次启用 `triton` 内核, `page_size` 静默降级, 仅在 `nightly` 覆盖, CI 状态未闭环

关联脉络

- PR #32093 [XPU] Adapt device agnostic API usage: 同属 XPU 设备无关化改造线，都在把硬编码 cuda / nccl 等平台判断替换为设备无关判断；本 PR 的 is_xpu 门控是同一思路在 DFLASH 路径上的延续。
- PR #32382 [XPU] Make checkpoint_engine worker device-agnostic: 同样在 worker 构建路径上补齐 XPU 平台分支并配套 XPU CI 与文档，与本 PR 在 dflash_worker_v2 / draft_worker_common 中的平台分支属于同一批 XPU 使能工作。
- PR #36631 [Sampling] Support sampling masks with overlap scheduling: 与 DFLASH verify 阶段的自定义采样掩码及 overlap 调度同属一条逻辑线；本 PR 在 _DFLASH_VERIFY_SKIP_CUSTOM_MASK_BACKENDS 中新增 XPUAttentionBackend、并覆盖 overlap / overlap-plan-stream 变体，都建立在该掩码处理机制之上。