

PR #42904 完整报告

vllm-project/vllm

[feature] [xgrammar] support `patternProperties`/`propertyNames`/`unevaluatedProperties` kw for object types

合并时间: 2026-09-15 12:58

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/42904>

执行摘要

- 一句话: xgrammar 后端放行简单 patternProperties/propertyNames, 并守卫会静默错编译的组合
- 推荐动作: 值得精读, 尤其是 vllm/v1/structured_output/backend_xgrammar.py 中守卫逻辑的分支条件, 和 tests/v1/structured_output/test_utils.py 的测试范式升级 (grammar_accepts + 独立 fixture)。建议重点追问四件事: 一是布尔形式的 propertyNames / additionalProperties 是否也应纳入守卫; 二是运行环境的 xgrammar 版本下限是否已在镜像或依赖中锁定到 0.2.1; 三是 patternProperties 与 properties 共存时的拒绝是否应配合更友好的错误提示 (当前用户只能看到 VLLMValidationError); 四是上游 #749/#826 修复后由谁负责回访放宽守卫。若线上开启了结构化输出, 建议用真实的 MCP schema 集合回归一遍, 而不是只依赖单测。

功能与动机

PR body 明确目标是让 OpenAI 兼容端点在结构化输出引擎下覆盖更多 JSON Schema: 「Support `patternProperties` and `propertyNames` keywords for `object` types in JSON schema for xgrammar structured output backend. Makes openai-compatible endpoint more feature-complete by covering more JSON schema under a structured output engine. This is supported since `xgrammar>=0.2.1` by `mlc-ai/xgrammar#594`。」。base 版本对这两类关键字是一刀切拒绝, 用户带 MCP 风格 schema 的请求只能拿到 400 (显式 xgrammar) 或回退到 outlines (auto)。review 过程中 `arpera` 与 `vadiklyuti` 引用上游缺陷 (xgrammar#749、#826、#839、#841) 指出存在「检测通过但约束不生效」的过度放行, 动机因此从「打开开关」演化为「只打开安全的开关」。

实现拆解

1. 判定策略反转 (vllm/v1/structured_output/backend_xgrammar.py): 删除 `any(key in obj for key in ("patternProperties", "propertyNames"))` 的一刀切拒绝, 改为删除后按组合重新判定。放行的是「无 `properties` 兄弟的单一 pattern `patternProperties`」与「不与属性类关键字冲突的独立 `propertyNames`」, 这也是本 PR 的功能本体; `validate_xgrammar_grammar` 仍负责把检测结果转成 `VLLMValidationError`, 从而驱动 `auto` 后端链式回退。

2. 新增三组守卫 (correctness 主线) : `propertyNames` 与 `properties` / `patternProperties` / dict 形式 `additionalProperties` / 非字面量 `true` 的 `unevaluatedProperties` 共存时判为不支持 (对应 xgrammar#826) ; `patternProperties` 与 `properties` 共存或 `pattern` 数 >1 时判为不支持 (xgrammar 把本应「同时满足」的合取编译成「满足其一即可」的析取) ; `propertyNames` 内部出现 `pattern/format + minLength/maxLength` 时判为不支持 (#749) 。为此把字符串分支的判定抽成 `_has_pattern_and_length_bounds`, 并显式套用到 `propertyNames` 上, 因为该关键字按惯例省略 `"type": "string"`, 会绕过原有字符串检查。
3. 健壮性与递归: `patternProperties` 的判长条件加了 `isinstance(obj.get("patternProperties"), dict)`, 避免 `patternProperties: null` 在守卫里抛 `TypeError`, 保证畸形 schema 能走到 `Grammar.from_json_schema()` 并被包装成 `VLLMValidationError`; `check_object` 的递归下钻保持不变, 嵌套对象里的同类组合同样被检查。`unevaluatedProperties` 只放行字面量 `True` (缺省也视为 `True`) , 因为 JSON Schema 允许布尔值, `False` 与 schema 形式都被拒绝。
4. 测试配套 (`tests/v1/structured_output/test_utils.py`) : 新增 `grammar_accepts`, 用 `Grammar.from_json_schema + xgrammar.testing.is_grammar_accept_string` 把断言从「检测器说支持」升级为「编译出的语法真的约束住了」, 并为 `property_names_schema` / `pattern_properties_schema` 各配一个具名行为测试。原来共享的 `unsupported_object_schemas` 拆成 `unsupported_property_names_combinations` 与 `unsupported_pattern_properties_combinations`, `supported_schema` 更名 `supported_frankenstein_schema` 并加上「禁止继续叠加关键字」的显式注释。
5. 回退路径与 CI 配套 (`tests/v1/structured_output/test_validation.py`) : `auto` 回退用例原先用「纯 `patternProperties`」, 该 schema 现在被 xgrammar 接受、不再触发回退, 于是换成 `properties + patternProperties` (xgrammar 与 guidance 都拒绝, 落到 outlines) , 修掉 `computer-cpu-v1-others` 的失败。依赖侧方面, 功能依赖 `xgrammar>=0.2.1` (mlc-ai/xgrammar#594) , 但最终 diff 只有 3 个文件、未见 requirements 变更, 版本下限需由环境 / 其他 PR 保证。

关键文件:

- `vllm/v1/structured_output/backend_xgrammar.py` (模块 结构化输出; 类别 source; 类型 core-logic; 符号 `_has_pattern_and_length_bounds`, `has_xgrammar_unsupported_json_features`, `check_object`) : 唯一的生产代码改动: 把对象关键字的「一刀切拒绝」改写为按组合放行 + 守卫, 并新增 `_has_pattern_and_length_bounds` 复用字符串长度边界判定, 直接决定哪些 JSON Schema 能进入 xgrammar 后端。
- `tests/v1/structured_output/test_utils.py` (模块 输出校验; 类别 test; 类型 test-coverage ; 符号 `grammar_accepts`, `unsupported_property_names_combinations`, `unsupported_pattern_properties_combinations`, `supported_frankenstein_schema`) : 测试范式升级的主战场: 引入 `grammar_accepts` 做行为断言, 把共享 schema 拆成独立 fixture, 并把新增的拒绝组合逐个落到 `unsupported_property_names_combinations` / `unsupported_pattern_properties_combinations`。
- `tests/v1/structured_output/test_validation.py` (模块 后端回退; 类别 test; 类型 test-coverage; 符号 `test_auto_backend_falls_back_on_unsupported_schema`) : 体现

auto 后端回退矩阵的语义变化：纯 patternProperties 现在被 xgrammar 接受、不再触发回退，因此回退用例必须换成 properties + patternProperties。

关键符号：has_xgrammar_unsupported_json_features,
_has_pattern_and_length_bounds, check_object, validate_xgrammar_grammar,
grammar_accepts, test_unsupported_json_features_by_type,
test_supported_json_features, test_property_names_constrains_keys,
test_pattern_properties_constrains_keys_and_values,
test_auto_backend_falls_back_on_unsupported_schema

关键源码片段

vllm/v1/structured_output/backend_xgrammar.py

唯一的生产代码改动：把对象关键字的「一刀切拒绝」改写为按组合放行 + 守卫，并新增 `_has_pattern_and_length_bounds` 复用字符串长度边界判定，直接决定哪些 JSON Schema 能进入 xgrammar 后端。

```
# vllm/v1/structured_output/backend_xgrammar.py
```

```
def _has_pattern_and_length_bounds(schema: dict[str, Any]) -> bool:
    # 抽出的公共判定：字符串上同时出现「生成式约束」(pattern/format) 与长度边界时,
    # xgrammar 只编译前者并静默丢弃 minLength/maxLength，语法与长度关键字无关,
    # 所以这种组合必须整体判为不支持，而不是悄悄放宽。
    return ("pattern" in schema or "format" in schema) and (
        "minLength" in schema or "maxLength" in schema
    )
```

```
def has_xgrammar_unsupported_json_features(schema: dict[str, Any]) -> bool:
    """Check if JSON schema contains features unsupported by xgrammar."""
```

```
def check_object(obj: dict[str, Any]) -> bool:
    if not isinstance(obj, dict):
        return False

    # 数值 / 数组 / 未知 format 的判定保持不变，此处略
    if obj.get("type") in ("integer", "number") and ("multipleOf" in obj):
        return True
    if obj.get("type") == "array" and any(
        key in obj
        for key in ("uniqueItems", "contains", "minContains", "maxContains")
    ):
        return True
    if (
        obj.get("type") == "string"
        and "format" in obj
        and obj["format"] not in STRING_SUPPORTED_FORMATS
    ):
```

```

    return True

# 字符串分支改为复用 helper，行为不变
if obj.get("type") == "string" and _has_pattern_and_length_bounds(obj):
    return True

# propertyNames 本质上是一个 string schema，按惯例常常省略 "type"，
# 会绕过上面的字符串检查，所以这里显式再查一遍同样的组合。
if (
    obj.get("type") == "object"
    and isinstance(obj.get("propertyNames"), dict)
    and _has_pattern_and_length_bounds(obj["propertyNames"])
):
    return True

# FIXME: propertyNames 与 properties/patternProperties/additionalProperties/
# unevaluatedProperties 共存时 xgrammar 行为不正确。
# https://github.com/mlc-ai/xgrammar/issues/826
# 注意 unevaluatedProperties 允许布尔值，因此只放行字面量 True（缺省也视为
# True），schema 形式与 False 一律拒绝；这里只看是否存在 propertyNames，
# 不检查其内部关键字，因为无论写 pattern 还是 enum，都不会被正确执行。
if (
    obj.get("type") == "object"
    and "propertyNames" in obj
    and (
        "properties" in obj
        or "patternProperties" in obj
        or isinstance(obj.get("additionalProperties"), dict)
        or obj.get("unevaluatedProperties", True) is not True
    )
):
    return True

# FIXME: 多个 patternProperties、或 patternProperties 与 properties 共存时，
# xgrammar 把本应「同时满足」(conjunction) 的约束编译成「满足其一即可」的
# 分支，所以只放行「单一 pattern 且没有 properties 兄弟」的组合。
# isinstance 守卫确保 patternProperties: null 这类畸形 schema 能落到
# Grammar.from_json_schema() 并被转换成 VLLMValidationError，而不是在这里
# 抛 TypeError。
if (
    obj.get("type") == "object"
    and isinstance(obj.get("patternProperties"), dict)
    and ("properties" in obj or len(obj["patternProperties"]) > 1)
):
    return True

# 递归下钻，保证嵌套对象 / 数组里的同类 schema 同样被检查
for value in obj.values():
    if isinstance(value, dict):

```

```

        if check_object(value):
            return True
    elif isinstance(value, list):
        for item in value:
            if isinstance(item, dict) and check_object(item):
                return True

    return False

return check_object(schema)

```

tests/v1/structured_output/test_utils.py

测试范式升级的主战场：引入 `grammar_accepts` 做行为断言，把共享 `schema` 拆成独立 fixture，并把新增的拒绝组合逐个落到 `unsupported_property_names_combinations` / `unsupported_pattern_properties_combinations`。

```

# tests/v1/structured_output/test_utils.py
import pytest
from xgrammar import Grammar
from xgrammar.testing import _is_grammar_accept_string

from vllm.v1.structured_output.backend_xgrammar import (
    has_xgrammar_unsupported_json_features,
)

pytestmark = pytest.mark.cpu_test

```

```

def grammar_accepts(schema: dict, text: str) -> bool:
    # 行为级断言：把 JSON Schema 编译成 grammar，再判断文本能否被接受。
    # 相比只调用 has_xgrammar_unsupported_json_features，它能暴露「检测器说支持、
    # 但编译出的语法其实没约束住」的假阳性。
    return _is_grammar_accept_string(Grammar.from_json_schema(schema), text)

```

```

@pytest.fixture
def unsupported_pattern_properties_combinations():
    return [
        # properties 与 patternProperties 命中同一个键时必须同时满足 (conjunction),
        # 但 xgrammar 编译成互斥分支，满足任一即可，所以这种 schema 必须判为不支持。
        {
            "type": "object",
            "properties": {"x": {"type": "string"}},
            "patternProperties": {"^x$": {"type": "integer"}},
        },
        # 同源问题：多个 patternProperties 之间也会退化成「或」语义。
        {
            "type": "object",
            "patternProperties": {

```

```

        "^[a-z]*$": {"type": "string"},
        "^[a-z]*z$": {"type": "integer"},
    },
},
]

```

```

@pytest.fixture
def pattern_properties_schema():
    # 单一 pattern、且不与 properties 共存 —— 这是本 PR 新放行的组合。
    return {
        "type": "object",
        "patternProperties": {"^grade_[0-9]+$": {"type": "integer"}},
    }

```

```

@pytest.fixture
def property_names_schema():
    return {"type": "object", "propertyNames": {"pattern": "^[a-z_]+$"}}

```

```

def test_property_names_constrains_keys(property_names_schema):
    # 值类型不受 propertyNames 约束，键名必须落在 pattern 内。
    assert grammar_accepts(property_names_schema, '{"score": 5}')
    assert grammar_accepts(property_names_schema, '{"score": "seven"}')
    assert not grammar_accepts(property_names_schema, '{"Score": 5}') # 大写键不匹配
    assert not grammar_accepts(property_names_schema, '{"score_1": 5}') # 后缀数字不匹配

```

```

def test_pattern_properties_constrains_keys_and_values(pattern_properties_schema):
    assert grammar_accepts(pattern_properties_schema, '{"grade_1": 5}')
    assert not grammar_accepts(pattern_properties_schema, '{"other": 5}') # 键不匹配 pattern
    assert not grammar_accepts(pattern_properties_schema, '{"grade_1": "five"}') # 值类型不符

```

同一文件里，原先共享的 `supported_schema` 被重命名为 `supported_frankenstein_schema`，并加了一条面向后续 PR 的约束（这正是 review 中争议最大的一条）：

```

@pytest.fixture
def supported_frankenstein_schema():
    # IMPORTANT(arpera):
    # 不要在这里继续叠加关键字！这个 schema 已经足够拥挤，新关键字可能对编译出的
    # grammar 完全无影响而测试依旧通过。新关键字请使用独立 fixture 加具名测试。
    return {
        # name / age / email / scores / car_type / address 等原有字段保持不变
        "type": "object",
        "minProperties": 1,
        "maxProperties": 100,
    }

```

`tests/v1/structured_output/test_validation.py`

体现 **auto** 后端回退矩阵的语义变化：纯 `patternProperties` 现在被 `xgrammar` 接受、不再触发回退，因此回退用例必须换成 `properties + patternProperties`。

```
# tests/v1/structured_output/test_validation.py

@pytest.mark.parametrize(
    "schema, expected_backend",
    [
        # multipleOf 仍被 xgrammar 拒绝，auto 回退到 guidance。
        (
            {
                "type": "object",
                "properties": {"n": {"type": "integer", "multipleOf": 2}},
            },
            "guidance",
        ),
        # properties + patternProperties 对 xgrammar（本 PR 新增的守卫）和 guidance
        # 都不支持，因此只能落到 outlines。
        # 原先这里用的是「纯 patternProperties」schema，本 PR 放行该组合后它不再触发
        # 回退，断言语义失效，故替换为当前的冲突组合。
        (
            {
                "type": "object",
                "properties": {"a": {"type": "string"}},
                "patternProperties": {"^a$": {"type": "string"}},
            },
            "outlines",
        ),
    ],
)

def test_auto_backend_falls_back_on_unsupported_schema(schema, expected_backend):
    """`auto` falls back on rejection, so it must catch what the validators raise."""
    params = SamplingParams(structured_outputs=StructuredOutputsParams(json=schema))
    params._validate_structured_outputs(
        _StubModelConfig(is_diffusion=False),
        StructuredOutputsConfig(backend="auto"),
        tokenizer=object(),
    )
    assert params.structured_outputs._backend == expected_backend
```

评论区精华

1. 测试冗余与覆盖缺口：`gemini-code-assist[bot]` 指出 `patternProperties` 用例命中已在 `properties` 中声明的键、属无效覆盖，并缺 `propertyNames` 用例；作者据此在 `supported_schema` 里加了 `student` 子对象，但这个子对象随后被推翻。
2. 「frankenstein schema」范式争议（PR 中最有价值的一条）：`arpera` 用 repro 脚本证明「`propertyNames` in this particular scheme does not affect on resulting grammar, so we do not test here behavior of `propertyNames`」，并主张「it is better to create

multiple schemas that check only one thing at a time」，因为拥挤 schema 会让新关键字被遮蔽而测试仍绿；`coderabbitai[bot]` 也从一致性角度指出 `propertyNames` + `patternProperties` 共存会让检测器返回 True、使 `test_supported_json_features` 失败。最终作者删除 `student` (commit `64d0e6e`)，拆出独立 fixture + 具名行为测试。

3. 三处 over-acceptance 的守卫：`vadiklyutiy` 转述 AI 评审给出三个具体反例（`propertyNames` 对具名 `properties` 不生效、`patternProperties` 合取被编译成析取、`propertyNames` 丢弃兄弟 `additionalProperties` 值 schema），作者确认全部成立并落地守卫，同时指出 AI 建议过窄：「`unevaluatedProperties` also admits boolean values by JSON Schema rulebook, so the only acceptable value for `unevaluatedProperties` is `true` for now」以及「item 1 applies not only to `pattern` but also `enum`... so we do not inspect what `propertyNames` contains」。
4. 测试该放哪一侧：作者提问是否该把新测试放到 xgrammar 仓库，`arpera` 主张留在 vLLM：「Xgrammar do not react swift enough for bug fixes that people try to submit them」，并列出被搁置的 #796/#839/#841；作者回应「Got it, I have no objection but just wanted a double check.」
5. 真实 E2E 佐证：`yzong-rh` 用 Qwen3.5 做了冒烟测试，main 上稳定失败（`VLLMValidationError + Regex compilation timed out after 5s`），本 PR 后 `patternProperties` 与 `propertyNames` 各 50/50 全部有效。
6. 未闭合分歧：`coderabbitai[bot]` 认为 `pattern_properties_schema` 需要补 `additionalProperties: false` 才能让 `assert not grammar_accepts(..., '{"other": 5})` 成立，但最终 fixture 未修改，讨论中也没有明确结论。
- 共享「frankenstein」schema 上叠加关键字会掩盖失效 (testing)：删除 `student` 字段（commit `64d0e6e`），把共享 schema 更名为 `supported_frankenstein_schema` 并加 IMPORTANT 注释禁止继续叠加，新增 `property_names_schema` / `pattern_properties_schema` 独立 fixture 与具名行为测试。
- 三处 over-acceptance 必须补守卫 (correctness)：在 `has_xgrammar_unsupported_json_features` 中新增 `propertyNames` 冲突守卫与 `patternProperties` 组合守卫，并把对应反例写成 `unsupported_*` fixture (commit `fbb1bfc`)。
- `patternProperties` 为 null 时守卫自身抛 `TypeError` (correctness)：判长条件改为 `isinstance(obj.get("patternProperties"), dict) and len(...) > 1`，畸形 schema 重新交回 xgrammar 处理 (commit `fe5417f`)。
- `pattern_properties_schema` 是否要加 `additionalProperties: false` (testing)：最终 fixture 未加该字段，讨论中也没有明确结论；新引入的 `grammar_accepts` 行为断言实际表明 xgrammar 的 `patternProperties-only` 分支会拒绝未匹配键，与 CodeRabbit 的判断相反，属于未闭合分歧。
- 回归测试放 vLLM 还是上游 xgrammar (design)：确认在 vLLM 侧保留针对 xgrammar 的行为级回归测试，作者无异议。
- auto 后端回退用例因功能放行而失效 (testing)：把回退用例换成 `properties` + `patternProperties` (xgrammar 与 guidance 都拒绝，落到 outlines)，并在注释中说明原因 (commit `1a3174fa`)。

风险与影响

- 风险:

1. 黑名单式守卫存在漏检面（核心风险）：新逻辑是「枚举已知的坏组合」，而非「枚举已知的好组合」。例如 `{"type": "object", "propertyNames": false}` 这类布尔 `propertyNames`、`propertyNames + 布尔 additionalProperties: false`（守卫只拒 dict 形式）、以及 `unevaluatedProperties` 在没有 `propertyNames` 时与 `properties` 共存，都不会被判为不支持，但 `xgrammar` 是否真正执行这些约束没有行为测试覆盖。一旦上游静默丢约束，`vLLM` 会在没有报错的情况下产出违反用户 schema 的 JSON，这是结构化输出中最难排查的失败模式。
1. 依赖版本下限未在 diff 内固定：功能依赖 `xgrammar>=0.2.1`（[mlc-ai/xgrammar#594](#)），但变更文件只有 3 个、未见 `requirements/` 镜像改动。若实际运行环境的 `xgrammar` 更旧，`vLLM` 侧守卫已经放行 `patternProperties/propertyNames`，上游却仍会丢约束，等价于静默放行。
2. 测试依赖私有 API：`tests/v1/structured_output/test_utils.py` 使用 `xgrammar.testing._is_grammar_accept_string`（下划线私有函数）与 `Grammar.from_json_schema`。`xgrammar` 升级改签名或改名会直接让 CI 崩，属于测试侧耦合。
3. `auto` 后端的归属变化：同一个 schema 从「回退到 outlines」变成「由 `xgrammar` 服务」，`grammar` 实现不同会带来输出风格、编译耗时与生成长度的差异；`test_auto_backend_falls_back_on_unsupported_schema` 的改写正是这条路径仍在演进的证据。
4. 守卫与上游修复的同步成本：所有拒绝分支都依赖注释里的 `FIXME` 链接（[#826/#749](#)），但代码里没有版本条件判断，`xgrammar` 修复后不会自动放宽，需要人工回访清理，否则会长期误拒合法 schema。
5. 行为断言与上游语义绑定：`assert not grammar_accepts(pattern_properties_schema, '{"other": 5})` 依赖 `xgrammar` 的 `patternProperties-only` 分支会拒绝未匹配键；这与 `CodeRabbit` 的判断相反且未在讨论中收口，语义漂移时测试会成为噪声源。- 影响：用户侧：OpenAI 兼容端点（含工具调用 /MCP 场景）中带 `patternProperties` 或 `propertyNames` 的 JSON Schema 请求，从「显式 `xgrammar` 直接 400、`auto` 回退 outlines」变为「由 `xgrammar` 用语法原生约束」。`yzong-rh` 的 E2E 显示 main 上必然失败、本 PR 后 50/50 全部通过，属于可见的功能补齐。

系统侧：改动只落在 `xgrammar` 后端的校验层（`has_xgrammar_unsupported_json_features` 及其调用方 `validate_xgrammar_grammar`），不触及调度、KV cache、模型执行路径；但 `auto` 后端的回退矩阵发生变化，会间接改变同一请求的 `grammar` 实现与性能特征。`outlines/guidance/lm-format-enforcer` 行为不受影响。

团队侧：确立了三条可复用的工程约定——「`vLLM` 侧保留 `xgrammar` 回归测试（上游响应慢）」「一个关键字一个 fixture + 具名行为测试，禁止堆叠共享 schema」「对上游缺陷用 `FIXME` 链接 + 守卫，而不是静默放行」。这些约定会影响后续所有结构化输出相关的 PR。

- 风险标记：黑名单式守卫易漏检，上游静默丢约束，依赖私有测试 API，`auto` 回退行为变更，版本下限未在 diff 内固定

关联脉络

- PR #48200 [Refactor] StructuredOutputManager x Speculative Decoding Refactor: 同一个结构化输出子系统 (vllm/v1/structured_output/ 与 tests/v1/structured_output/) , 并同样涉及「校验 / 约束起点契约」的演进, 可对照理解本次对 xgrammar 支持面判定的定位。
- PR #56408 [Frontend] Use XGrammar schema constraints for DeepSeek V4.1: 同样围绕 XGrammar 的 schema 约束能力做前后端打通 (structural tag 注册与回退), 与本 PR 在「哪些 schema 交给 xgrammar 强约束、其余回退」的策略上同源。