

PR #38690 完整报告

sgl-project/sglang

[Feat][Responses API] Support custom tools, encrypted reasoning replay, developer tier and model validation

合并时间: 2026-09-12 21:29

原文链接: <http://prhub.com.cn/sgl-project/sglang/pull/38690>

执行摘要

- 一句话: Responses API 补齐 custom 工具、加密推理回放与模型校验
- 推荐动作: 值得精读, 尤其是三类人: 维护 OpenAI 兼容入口的工程师 (应重点看 `responses_adapters.py` 的 shim 设计与 `serving_responses.py` 的 `_validate_model` / `effective_tool_choice` / `_terminal_stream_event` 三处入口校验)、做推理 / 回放功能的工程师 (看 `encode_reasoning_state` 的「编码非加密」定位与 `msg_store` 的 `_response_input_history` 结构)、以及 gpt-oss / Harmony 相关开发者 (看 `harmony_utils.py` 的 `channel↔phase` 双向映射)。建议重点关注的设计决策: ① 用「单字符串属性的 function 工具」承载 custom 工具, 并用增量反转义保持 delta 与终值一致; ② 具名 `tool_choice` 与 `required` 共享一条强制解析路径, 未声明的名字在生成前 400 而非生成后丢弃; ③ 用「缓冲到 phase 可知」换 `added/done/final` 三处 phase 一致; ④ 主动把 `compact`、真 grammar 强制、真加密列为 out of scope 而不是半实现。阅读时建议优先补看 PR body 的症状表与 issue 里 JustinTong0323 的验证记录, 再对照 diff 验证每条症状的落点。

功能与动机

PR body 用一张症状一根因表说明了动机: Codex-class 客户端在 `/v1/responses` 上暴露了一连串翻译层缺口——声明 `custom` 工具得不到调用且 `tool_choice="required"` 直接 400; `include=["reasoning.encrypted_content"]` 只被校验却不填充, `store=false` 客户端无法回放推理轨迹; `developer` 消息折叠成 `system` 后层级丢失被后续 `user` 消息压过; 未知 `model` 仍用已加载模型回答并返回 HTTP 200; 具名 `tool_choice` 生成了合法工具 JSON 却被当普通文本返回; 流式与全量输出在 `required/native parser` 上判定不一致; 截断 / 中止流恒以 `response.completed` 结束, 违反 `response.incomplete` / `response.failed` 契约; `previous_response_id` 续写丢失工具调用与推理, 新 `instructions` 还会与上一轮拼接。作者明确说明这些都是 request/response translation concerns, 不涉及调度、kernel 或采样。

实现拆解

1. 新增适配层 `python/sglang/srt/entrypoints/openai/responses_adapters.py` (+215, 全新文件): 把两个 chat-completions 路径无法表达的概念收拢到一个纯函数模块。
`custom_tool_parameters()` 把 custom 工具伪装成只含 `input: string` 的 function 工具,
`custom_tool_description()` 把 `format.type == "grammar"` 的 `syntax/definition` 折进工具

描述（只描述、不强制，因为 payload 位于模型自己的 tool-call 信封内，约束解码无法只作用于它）；`encode_custom_tool_input()` / `decode_custom_tool_input()` 负责 payload 与 shim arguments 的双向转换，后者对裸文本与非 input 单键对象做了兜底；`decode_custom_tool_input_prefix()` 配合 `_payload_start()` / `_unicode_escape()` 在半截 arguments JSON 上做增量反转义（含 `\uXXXX` 代理对），保证流式 delta 与最终 item 的 input 严格一致。此外 `label_developer_content()` 给 developer 块加 Developer instructions: 前缀，`encode_reasoning_state()` / `decode_reasoning_state()` 用 `sglang-reasoning-v1`. 前缀 + `zlib` + `urlsafe base64` 承载推理轨迹。

2. 扩展协议模型 `python/sglang/srt/entrypoints/openai/protocol.py` (+43/-8) :
ResponseTool 增加 format 字段描述 custom 工具输入格式；新增 ResponseInputMessageParam（继承 EasyInputMessageParam, total=False）与 ResponseOutputMessage（继承 openai 的 ResponseOutputMessage）各自带 phase: Optional[Literal["commentary", "final_answer"]], 并接入 ResponseInputOutputItem 与 ResponsesResponse.output 两处 Union, 绕开被 pin 的 SDK 模型；新增 `is_include_encrypted_reasoning()`; `effective_tool_choice()` 从只认具名 function 扩展为 function / custom, 其余对象形式一律退化 auto; ChatCompletionMessageGenericParam 也加上 phase, 使 phase 能在 chat 内部消息上流转。
3. 服务端主流程 `python/sglang/srt/entrypoints/openai/serving_responses.py` (+701/-631) :
`_known_model_names()` 汇总 served_model_name 与 LoRA registry adapter 名, `_validate_model()` 对未知 model 返回 404 OpenAI-style error (None、served 名、adapter 名、base-model:adapter 均可通过) ; `create_responses()` 在生成前完成 model 校验、required 接受 function/custom、具名 choice 未声明则 400; 强制解析路径在全量与流式之间共享 `_tool_parser_owns_format()` 判定 (统一 supports_structural_tag 与 parses_required_natively) ; 终止事件改由 `_terminal_stream_event()` / `_error_from_finish_reason()` 依据最终 status 产出 completed/incomplete/failed; 存储历史改为 `messages[2:]` (Harmony, 丢弃按请求重建的前缀) 或 `_response_input_history(request)` 加完整 output item, 回放复用与客户端一致的 normalizer; `_make_reasoning_item()` 在 include 命中时填充 encrypted_content, `_make_tool_call_item()` / `_flatten_tool_output()` 支撑 custom_tool_call 与非流式工具结果归并。
4. Harmony 桥接 `python/sglang/srt/entrypoints/harmony_utils.py` (+102/-20) : 文本分片过滤加入 output_text, 修复 assistant 文本在回放时被丢弃; reasoning 分支由「断言 content 长度为 1」改为优先 summary/content、为空再回落 `decode_reasoning_state(encrypted_content)`; `function_call_output/custom_tool_call_output` 统一处理并支持 list 形态 output 拼接, custom_tool_call 回放时用 `encode_custom_tool_input()` 重新包信封; `parse_output_message()` / `parse_remaining_state()` 增加 python recipient → ResponseCodeInterpreterToolCall、commentary 无 recipient → message (phase=commentary)、final → phase=final_answer, 并把 `split(".")[1]` 换成 `removeprefix("functions.")` 避免函数名含点被截断。

5. 测试与 CI 配套：新增 test/registered/unit/entrypoints/openai/test_responses_custom_tools.py (+442, 含 encode/decode、增量前缀解码、glm47 真实线格式、回放与流式 delta 重建)；test_serving_responses_stream.py (+322/-2) 新增终态事件、native parser 一致性、Harmony 流式 item 生命周期与截断 item 收敛用例；test_serving_responses.py (+215/-5) 覆盖存储回放等价于客户端自带上文、instructions 不跨请求继承、phase 保持与 Harmony 强制 choice 的 400 文案；test/.../openai/utils.py 给 MockTokenizerManager 补 SERVED_MODEL_NAME = "x"、served_model_name、lora_registry = None, 否则 model 校验上线后所有单测都会 404。测试统一 register_cpu_ci(est_time=10, suite="base-a-test-cpu"), 进入 CPU CI 基线套件。

关键文件：

- python/sglang/srt/entrypoints/openai/serving_responses.py (模块 响应服务层；类别 source；类型 core-logic；符号 _known_model_names, _validate_model, create_responses, _make_reasoning_item)：Responses API 的主服务层，承载 model 校验、tool_choice 前置拒绝、终态 SSE 事件、状态化回放与 phase 保持，是本 PR 逻辑最密集、回归面最大的文件 (+701/-631)。
- python/sglang/srt/entrypoints/openai/responses_adapters.py (模块 响应适配层；类别 source；类型 core-logic；符号 custom_tool_parameters, custom_tool_description, custom_tool_names, encode_custom_tool_input)：全新适配层，集中承载 custom 工具 shim、增量 JSON 反转义、developer 层级标签与 reasoning 状态编解码；被 serving_responses.py 与 harmony_utils.py 共同依赖，是本 PR 的设计核心。
- python/sglang/srt/entrypoints/openai/harmony_utils.py (模块 消息桥接层；类别 source；类型 core-logic；符号 parse_response_input, parse_output_message, parse_remaining_state)：Harmony 回放与输出的桥接层：修复 assistant 文本被过滤、encoded-only 推理无法回放、custom_tool_call 缺回放路径，并把 message channel 双向映射到 phase。
- python/sglang/srt/entrypoints/openai/protocol.py (模块 协议模型；类别 source；类型 data-contract；符号 ResponseInputMessageParam, ResponseOutputMessage, is_include_encrypted_reasoning, effective_tool_choice)：线格式模型的契约来源：新增 format 与 phase 字段、is_include_encrypted_reasoning(), 并调整 effective_tool_choice() 与两处 Union，决定了下游所有翻译逻辑的输入类型。
- test/registered/unit/entrypoints/openai/test_responses_custom_tools.py (模块 单元测试；类别 test；类型 test-coverage；符号 _custom_request, CustomToolAdapterTestCase, test_payload_survives_encode_decode, test_prefix_decode_tracks_a_growing_buffer)：新增 442 行 custom 工具测试，覆盖 encode/decode 往返、增量前缀解码的单调性、shim 工具形态、具名与 required 的全量 / 流式一致性，以及真实 glm47 线格式的解析。
- test/registered/unit/entrypoints/openai/test_serving_responses_stream.py (模块 单元测试；类别 test；类型 test-coverage；符号 test_truncated_and_aborted_streams_have_matching_terminal_events, test_required_native_parser_matches_full_response, HarmonyStreamLifecycleTestCase, test_truncated_harmony_arguments_close_the_em

itted_item)：流式契约的主回归网：新增终态事件 (incomplete/failed)、native parser 全量 - 流式一致、phase 透传与 Harmony 流式 item 生命周期用例。

- test/registered/unit/entrypoints/openai/test_serving_responses.py (模块 单元测试; 类别 test; 类型 test-coverage; 符号 test_stored_tool_turn_matches_client_replay_without_old_instructions, test_harmony_instructions_are_rebuilt_for_each_request, test_harmony_replays_output_text_and_encoded_reasoning, test_harmony_replays_dict_tool_calls_and_results_in_one_input)：非流式侧回放与 phase 的回归网：断言存储回放等价于客户端自带上文、旧 instructions 不继承、Harmony instructions 按请求重建与强制 choice 的 400 文案。
- test/registered/unit/entrypoints/openai/utils.py (模块 单元测试; 类别 test; 类型 test-support; 符号 MockTokenizerManager.SERVED_MODEL_NAME)：测试夹具适配：MockTokenizerManager 补上 served_model_name 与 lora_registry, 否则 model 校验上线后整套 Responses 单测都会 404。

关键符号：_known_model_names, _validate_model, create_responses, _response_tools_to_chat_tools, _normalize_response_message_for_chat, _response_input_history, _make_reasoning_item, _make_tool_call_item, _error_from_finish_reason, _terminal_stream_event, _tool_parser_owns_format, _flatten_tool_output, responses_stream_generator, responses_full_generator, effective_tool_choice, is_include_encrypted_reasoning, custom_tool_parameters, custom_tool_description, encode_custom_tool_input, decode_custom_tool_input, decode_custom_tool_input_prefix, label_developer_content, encode_reasoning_state, decode_reasoning_state, parse_response_input, parse_output_message, parse_remaining_state

关键源码片段

python/sglang/srt/entrypoints/openai/serving_responses.py

Responses API 的主服务层，承载 model 校验、tool_choice 前置拒绝、终态 SSE 事件、状态化回放与 phase 保持，是本 PR 逻辑最密集、回归面最大的文件 (+701/-631)。

python/sglang/srt/entrypoints/openai/serving_responses.py (整理后的关键实现)

```
def _known_model_names(self) -> set[str]:
    """调用方可以寻址的 model id, 口径对齐 ``/v1/models``。"""
    names = {self.tokenizer_manager.served_model_name}
    # LoRA adapter 也算合法 model id; registry 不存在时静默跳过。
    registry = getattr(self.tokenizer_manager, "lora_registry", None)
    if registry is not None:
        names.update(registry.get_all_adapters().keys())
    return names
```

```
def _validate_model(self, model: Optional[str]) -> Optional[ORJSONResponse]:
    """拒绝未知 ``model``，与 Responses API 的行为对齐。
```

``None`` 表示「用当前已加载的模型」；LoRA adapter 既可以按名字寻址，也可以用 ``base-model:adapter`` 形式。

```
"""
```

```
if model is None:
    return None
base_model, _ = self._parse_model_parameter(model)
known = self._known_model_names()
if model in known or base_model in known:
    return None
return self.create_error_response(
    message=f"The model '{model}' does not exist",
    err_type="invalid_request_error",
    status_code=HTTPStatus.NOT_FOUND,
    param="model",
)
```

```
async def create_responses(self, request, raw_request=None):
```

```
    if not self.tokenizer_manager:
        return self.create_error_response("Model not loaded")
```

```
# 未知 model 直接 404，而不是「静默用已加载模型回答 + HTTP 200」。
```

```
model_error = self._validate_model(request.model)
```

```
if model_error is not None:
```

```
    return model_error
```

```
# tool_choice="required" 现在接受 function 与 custom 两类可强制的工具。
```

```
if request.tool_choice == "required" and not any(
    tool.type in ("function", "custom") for tool in (request.tools or [])
):
```

```
    return self.create_error_response(
        'tool_choice="required" requires at least one tool with '
        'type="function" or type="custom"; other built-in tool types '
        "cannot be forced."
    )
```

```
# 具名 tool_choice 收敛成 {"type": "function", "name": ...};
```

```
# 未声明的名字在生成前就 400，避免「生成了合法工具 JSON 却当普通文本返回」。
```

```
tool_choice = request.effective_tool_choice()
```

```
if isinstance(tool_choice, dict) and not any(
    tool.type in ("function", "custom") and tool.name == tool_choice["name"]
    for tool in request.tools or []
):
```

```
    return self.create_error_response(
        f"Tool {tool_choice['name']}!r is not declared in tools",
        param="tool_choice",
    )
```

```
# ... 省略 logprobs / json_schema / web_search 的既有校验 ...
```

```

# 存储历史: 保留原始输入 + 完整 output item,
# 回放时走与客户端自带上文完全相同的 normalizer。
if request.store:
    self.msg_store[request.request_id] = (
        messages[2:] # Harmony: 丢弃按请求重建的 system/developer 前缀
        if self.use_harmony
        else self._response_input_history(request)
    )

```

python/sglang/srt/entrypoints/harmony_utils.py

Harmony 回放与输出的桥接层: 修复 assistant 文本被过滤、encoded-only 推理无法回放、custom_tool_call 缺回放路径, 并把 message channel 双向映射到 phase。

python/sglang/srt/entrypoints/harmony_utils.py (整理后的关键分支)

```

def parse_response_input(response_msg, prev_responses):
    # ... 前面的 message / developer 分支省略 ...
    if response_msg["type"] in ("function_call_output", "custom_tool_call_output"):
        call_id = response_msg["call_id"]
        call_response = None
        for previous in reversed(prev_responses):
            # 历史里既有 TypedDict 也有 pydantic 对象, 先统一成 dict 再比对。
            if not isinstance(previous, dict):
                previous = previous.model_dump()
            if (
                previous.get("type") in ("function_call", "custom_tool_call")
                and previous.get("call_id") == call_id
            ):
                call_response = previous
                break
        if call_response is None:
            raise ValueError(f"No call message found for {call_id}")
        output = response_msg.get("output", "")
        if isinstance(output, list):
            # custom_tool_call_output 的 output 允许是 output_text 分片列表。
            output = "".join(
                part.get("text", "") for part in output if isinstance(part, dict)
            )
        msg = Message.from_author_and_content(
            Author.new(Role.TOOL, f"functions.{call_response['name']}"), output
        )
    elif response_msg["type"] == "reasoning":
        # 优先 summary / content; 两者皆空时回落到 encrypted_content 的编码态。
        text = ""
        for field in ("summary", "content"):
            text = "\n".join(
                part.get("text", "")
                for part in response_msg.get(field) or []
                if isinstance(part, dict)
            )

```

```

    )
    if text:
        break
    if not text:
        text = decode_reasoning_state(response_msg.get("encrypted_content")) or ""
    msg = Message.from_role_and_content(Role.ASSISTANT, text).with_channel(
        "analysis"
    )
elif response_msg["type"] in ("function_call", "custom_tool_call"):
    # custom_tool_call 的 input 要重新包成 shim 的 JSON 参数再喂回模型。
    arguments = (
        encode_custom_tool_input(response_msg.get("input") or "")
        if response_msg["type"] == "custom_tool_call"
        else response_msg.get("arguments") or "{}"
    )
    if isinstance(arguments, dict):
        arguments = orjson.dumps(arguments).decode()
    msg = Message.from_role_and_content(Role.ASSISTANT, arguments)
    msg = (
        msg.with_channel("commentary")
        .with_recipient(f"functions.{response_msg['name']}")
        .with_content_type("json")
    )

def parse_output_message(message):
    # ... web_search / python 等 recipient 分支省略 ...
    if message.recipient is not None and message.recipient.startswith("functions."):
        # 用 removeprefix 取代 split(".")[1]，避免函数名里带点被截断。
        function_name = message.recipient.removeprefix("functions.")
        # ... 逐个 content 生成 ResponseFunctionToolCall ...
    elif message.channel == "final" or (
        message.channel == "commentary" and message.recipient is None
    ):
        # final channel 是 final_answer; commentary 但无 recipient 的正文
        # 按 Codex 语义仍是 commentary 阶段。
        text_item = ResponseOutputMessage(
            phase="final_answer" if message.channel == "final" else "commentary",
            id=f"msg_{random_uuid()}",
            content=contents,
            role=message.author.role,
            status="completed",
            type="message",
        )

```

评论区精华

review 侧信息很薄: `review_comments_count = 0`, 唯一一条审核是 JustinTong0323 的 **APPROVED** (无正文), 最终由 ShangmingCai 合并; 讨论实际发生在 PR body 和 issue 评

论里。

- JustinTong0323 在 issue 评论贴出后 7 个 follow-up commit (0c08092..cc5b0e2) 的验证记录：> Unit tests (pinned openai==2.6.1, CPU) : 392 passed, 137 subtests passed; Codex conformance suite 27 cases: 24 PASS / 1 FAIL / 2 SKIP; FAIL 是 compaction_replay (POST /v1/responses/compact 不存在，故意不在本 PR 范围)，2 个 SKIP 是模型每次只发一个调用、并行路径根本没被跑到 (model behavior, not a server failure)；真实 codex-cli 0.153.4 会话 31/31 请求 HTTP 200，成功从零写出带 32 个单测的终端贪吃蛇并修好一个计算器解析器。
- PR body 主动划出 out of scope: POST /v1/responses/compact、custom 工具的真实 grammar 强制 (需要 staged generation)、真正的加密推理状态与服务端密钥管理、Harmony 下超过显式 400 的强制 tool_choice 以及 Harmony json_schema。
- commit abb8b3f 的 message 记录了一个明确的取舍：> Buffer text events when tools are enabled until the next tool call or terminal event resolves commentary versus final_answer. This delays text delivery for tool-capable requests but keeps added, done, and final output phases consistent.——即为了 phase 一致性主动牺牲工具请求的流式文本时延。
- 过程中出现 4 次 /rerun-failed-ci (均来自 JustinTong0323)，body 里贴出的 CI 状态为 Base :no_entry_sign:、Extra 与 AMD ROCm 均 :x:，说明验证靠反复重跑收敛而非一次绿。
- custom 工具的 grammar 只描述不强制 (design): 接受为当前限制并列入 out of scope (需要 staged generation 才能真正强制)，仅通过描述引导模型。
- encrypted_content 是编码而非加密 (security): 接受编码方案，但语义上必须被当作「对客户端不透明」而非安全边界；decode_reasoning_state() 对客户端传入 blob 无解压体积上限。
- phase 未知时缓冲 message SSE 事件 (performance): 采用缓冲方案，代价是工具请求首字延迟上升；测试以 test_final_output_preserves_text_tool_text_order 的 phase 断言锁定行为。
- Harmony 强制 tool_choice 改为显式 400 (correctness): 从「未实现异常」升级为可读的 400，行为更明确但仍不支持真正强制。
- 全量与流式共享 native 解析判定 (correctness): 已统一并补回归测试，流式与全量在工具名与 arguments 上逐字一致。
- 存储历史改为原始输入 + 完整 output item (design): 已解决，并作为契约写进测试；msg_store 值类型相应放宽为 list[dict]。
- 验证证据：单测 + Codex 一致性 + 真机 CLI (testing): 以「单测 + 一致性套件 + 真机会话 + 精度回归」四层证据支撑合并；唯一 FAIL 与两个 SKIP 均已在记录中说明原因。

风险与影响

- 风险：

1. 向后兼容破坏 (中高)：_validate_model() 让未知 model 从「用已加载模型回答 + HTTP 200」变成 404。虽然 None、served 名、adapter 名、base-model:adapter 都

能通过，但老客户端若传路由前缀或自定义别名会被直接拒绝；

`_known_model_names()` 还依赖 `tokenizer_manager.served_model_name` 与可选 `lora_registry`，这两个属性缺失会退化为只认 `served` 名。

2. `custom` 工具异常路径不一致（中）：`decode_custom_tool_input_prefix()` 强依赖参数 JSON 里出现 `"input": "`，测试里 `decode_custom_tool_input_prefix({'city': "Beijing"})` 断言为空串。若模型换用别的键名，流式场景会一个 `response.custom_tool_call_input.delta` 都不发，而最终 `item` 仍能通过 `decode_custom_tool_input()` 的单键兜底拿到 `payload`——同一响应内流式增量与终值语义不一致，客户端重建逻辑容易被误导。
3. `encrypted_content` 并非加密（中）：`encode_reasoning_state()` 是 `zlib + base64` 的纯编码，`decode_reasoning_state()` 对客户端传入的 `blob` 直接 `zlib.decompress()`。一方面名字暗示保密性而实际无密钥，容易被下游误当安全边界；另一方面解压客户端可控数据存在内存放大风险，代码里没有解压体积上限或输入长度检查。
4. 流式事件契约回归面大（中）：`Harmony` 流式改为由单一 `emitter` 统一持有 `item id / call id / output_index / sequence_number`，并要求终态响应复用已发出 `item`。`test_serving_responses_stream.py` 覆盖了 `split/coalesced` 消息与截断 `item`，但任何 `emitter` 与 `parser snapshot` 失配都会让依赖 `output_index` 的客户端错位，属于高敏感区。
5. 流式时延回退（低 - 中）：按 `commit abb8b3f` 的描述，启用 `tools` 的请求会把 `message SSE` 事件缓冲到下一个 `tool call` 或终态事件才能确定 `commentary / final_answer`，文本首字延迟上升；`text-only` 流不受影响。
6. 评审深度不足（流程风险）：`+2045/-666` 的大改动、`msg_store` 值类型由 `list[ChatCompletionMessageParam]` 放宽为 `list[dict]`、`ResponseInputOutputItem` 联合类型重排等结构性变化，只有一条无正文 `APPROVED`，0 条内联评论；好在配套单测（392 passed / 137 subtests）与真机验证记录补齐了部分信心。- 影响：用户侧：使用 `Codex CLI / Codex-class` 客户端且 `wire_api="responses"`、`store=false` 的场景直接受益——`custom` 工具可用、具名与 `required` 的 `tool_choice` 才会真正触发工具调用、推理轨迹可跨请求回放、截断流不再伪装成成功、`phase` 让 `commentary / final_answer` 可区分。使用 `gpt-oss (Harmony)` 的用户行为变化最大：`tool_choice != auto` 从 `NotImplementedError` 变成带解释的 400，`previous_response_id` 的 `system/developer` 前缀改为按请求重建，历史回放语义更接近规范但不再继承旧 `instructions`。系统侧：改动集中在 `sglang/srt/entrypoints/openai/` 与 `entrypoints/harmony_utils.py`，不动 `scheduler / kernel / 采样`，理论上不影响吞吐与显存，但引入了 `responses_adapters.py` 这一新的翻译层依赖，`serving_responses.py` 与 `harmony_utils.py` 都要从它导入，后续任何人改线格式都需要同步两处。团队侧：新增 442 行 `custom` 工具测试并注册到 `base-a-test-cpu`，CPU CI 时长与维护面积上升；`serving_responses.py` 单文件 ± 1300 行级别的重排会让后续 `git blame / 回溯` 变难。- 风险标记：核心协议路径变更，向后兼容破坏（未知 `model` 变 404），流式事件契约调整，编码非加密的语义风险，`custom` 工具异常路径不一致，无内联评审记录

关联脉络

- PR #38908 Fix gpt-oss RunAI streamer weight ownership: 同属 gpt-oss / Harmony 服务路径的质量修复；本 PR 在 harmony_utils.py 与 serving_responses.py 的 Harmony 分支上做了 channel↔phase 映射、回放与流式生命周期改造，两条线都在加固 gpt-oss 的 Responses 服务体验。
- PR #36411 [Perf] Optimize Qwen3-VL unique-image serving on H100: 同属 OpenAI 入口层 (entrypoints/openai/)，都改动了 serving 侧请求翻译与 serving_chat.py 相关链路；本 PR 复用了 chat serving 的 _parse_model_parameter 等入口设施。
- PR #39035 [Session] Fix session idle timeout after rejected requests: 同属 OpenAI 入口层的请求生命周期修复，都涉及「被拒绝的请求不应污染后续状态」这一类语义；本 PR 把未声明 tool_choice 与未知 model 提到生成前拒绝，思路与之相近。