Dify streaming, cancellation, and binary responses

This reference defines the Dify connector response paths after an action passes admission. Use it to select an execution mode and preserve uncertainty when a provider task cannot be proved stopped.

SECTION	Connectors
TYPE	Connector
PROTOCOL	AIP 1.0
SOURCE	docs/connectors/dify/reference/streaming-cancellation-and-binary-responses.md

The connector has three distinct response treatments: normalized SSE, bounded binary data, and buffered JSON or text. The catalogue, not an upstream `Content-Type` guess, selects the treatment.

Select the response path

Capability	Typed invocation	Synchronous invocation	Cancellation contract
App alias	Normalized SSE	Forced blocking JSON	Remote stop after task identity is known
`workflow.run_by_id`	Normalized SSE	Forced blocking JSON	Remote workflow stop
`workflow.event.stream`	Normalized SSE	Buffered response	Close the local subscription only
Other app operation	Buffered response	Buffered response	Not supported
Knowledge operation	Buffered response	Buffered response	Not supported

Typed invocation forces `response_mode: streaming` for an app alias and `workflow.run_by_id`. Synchronous invocation forces `response_mode: blocking` for the same execution routes.

`workflow.event.stream` is already an SSE route. Cancelling its AIP action must not infer a request to stop the underlying workflow.

Provider pipeline run and node-run endpoints may use SSE internally. Their frozen catalogue entries declare JSON responses, so this connector buffers them and does not advertise AIP streaming or cancellation.

Consume normalized SSE

The parser accepts LF or CRLF frame delimiters. It collects all `data` lines in a frame, joins them with newline characters, and parses the result as JSON.

Comment-only and field-only frames produce no AIP chunk. A literal `[DONE]` becomes a synthetic `message_end` event.

Every parsed event must contain an `event` string. The connector uses `data` as the event payload when present; otherwise, it retains the full provider object.

Map event kinds

Dify event	AIP stream kind	Special result behavior
`workflow_started`, `node_started`	Progress	None
`tool_call`	Tool	None
`agent_thought`	Thought	None
`message`	Data	Appends a non-empty `answer` delta
`human_input_required`	Pending approval	Records human-input data
`workflow_paused`	Pending approval	Terminal and records human-input data
`error`	Error	Produces a failed action result
`message_end`, `workflow_finished`	Done	Terminal
Any other event	Progress	Provider data remains available

An unsuccessful `workflow_finished` status produces a failed result even though the event is terminal. HTTP success therefore does not imply successful workflow completion.

The emitted sequence begins at zero and increases once per parsed event. Message deltas are also accumulated for the final result.

Recognize terminal outcomes

The terminal events are `message_end`, `workflow_finished`, and `workflow_paused`. An upstream close before any terminal event is an invalid stream.

When message deltas exist, the completed output contains the assembled answer and final event. Without deltas, the final event data becomes the output.

Recorded human-input data changes the final action outcome to requires-human. It remains attached with the final event rather than being flattened into a normal completion.

Enforce stream bounds

Bound	Frozen value or source
Complete response or stream	Configured maximum, 8 MiB by default
Maximum configurable response	64 MiB
Undelimited SSE frame	1 MiB
Emitted events	10,000
Accumulated answer	Configured response maximum

A transport chunk may contain several complete frames and may exceed 1 MiB. Only the undelimited remainder is subject to the frame bound.

The complete byte count includes every upstream transport chunk. A declared `Content-Length` above the configured maximum is rejected before body collection when that header is available.

Preserve durable cancellation evidence

The connector correlates an AIP action ID with three provider values:

```
TEXT
app_id + task_id + user
```

The first parsed event carrying `task_id`, either at the top level or under `data`, creates or replaces that durable mapping. The same `user` value used to start the action is stored because Dify stop routes require it.

A cluster-safe deployment uses the profile-state task store. The in-memory store is suitable only when restart and replica handoff do not need to preserve cancellation evidence.

Interpret cancellation certainty

State at cancellation	Connector action	Reported certainty
Request not yet admitted or no task ID observed	Interrupt locally	Remote stop unconfirmed
Durable task mapping exists	Call the mode-specific stop route	Confirmed after success or 404
Explicit `task_id` and valid `user` are supplied	Reconstruct task identity and call stop	Confirmed after success or 404
Event subscription capability	Close local response stream	No workflow-stop claim
Non-cancellable explicit operation	Reject cancellation	No provider mutation

The stop route is selected from the configured app mode. Workflow, completion, and chat-family apps use different provider paths.

A provider 404 is accepted as an idempotent stop outcome. It proves that the target task is no longer available at the stop route, not why it disappeared.

After confirmed stop, normal terminal completion, or provider-declared failure, the connector removes the durable mapping.

Handle stream failure cleanup

Malformed UTF-8, invalid JSON, a missing event name, exceeded bounds, and premature close fail stream processing. The connector then attempts to stop a known active task.

If cleanup itself cannot be confirmed, the returned stream error includes that uncertainty. Never translate this state into a successful cancellation or a safe retry decision.

Mutating app aliases and workflow runs remain unsafe to retry. An idempotency key controls replay identity but does not prove that an interrupted provider execution had no effects.

Decode binary envelopes

Exactly three frozen operations declare binary responses:

- `file.preview`
- `audio.synthesize`
- `document.download_zip`

The connector reads the body under the configured response bound and returns this envelope:

```
JSON
{
  "http_status": 200,
  "content_base64": "<base64 bytes>",
  "content_type": "application/octet-stream",
  "size_bytes": 128,
  "provider_request_id": "<optional x-request-id>"
}
```

`content_type` and `provider_request_id` can be absent. Consumers must decode `content_base64`, verify `size_bytes`, and apply a trusted media policy before opening or persisting the result.

The connector does not infer a filename or parse the binary payload. Base64 expansion also means the AIP output is larger than the provider body.

Interpret buffered non-binary responses

All other explicit operations return a bounded envelope containing `http_status`, `body`, `content_type`, and optional `provider_request_id`.

Valid JSON becomes the `body` value. Empty content becomes null. Non-JSON bytes become a lossy UTF-8 `text` field, so this fallback is diagnostic rather than a binary-safe transport.

Provider error responses are decoded under the same response bound. Preserve the mapped AIP failure and retry metadata instead of branching only on the embedded provider body.

Diagnose response failures

Observation	First decision
No chunks arrive	Verify typed invocation and an SSE-declared capability
Stream closes without completion	Treat the outcome as invalid and inspect cleanup certainty
Cancellation is unconfirmed	Look for durable task identity; do not replay the mutation
Cancellation stops event delivery but workflow continues	Confirm the capability is `workflow.event.stream`
Pipeline run has no AIP chunks	Its frozen response kind is JSON
Binary body appears as text	Verify the operation is one of the three binary catalogue entries
Response exceeds a bound	Reduce provider output or raise the configured limit within 64 MiB
Workflow ends with HTTP 200 and failed status	Use the action result, not HTTP status alone

Related documentation

- Dify connector overview
- Run Dify chat and workflows
- Dify workflows, streaming, and human input
- Dify connector configuration
- Errors and retry decisions