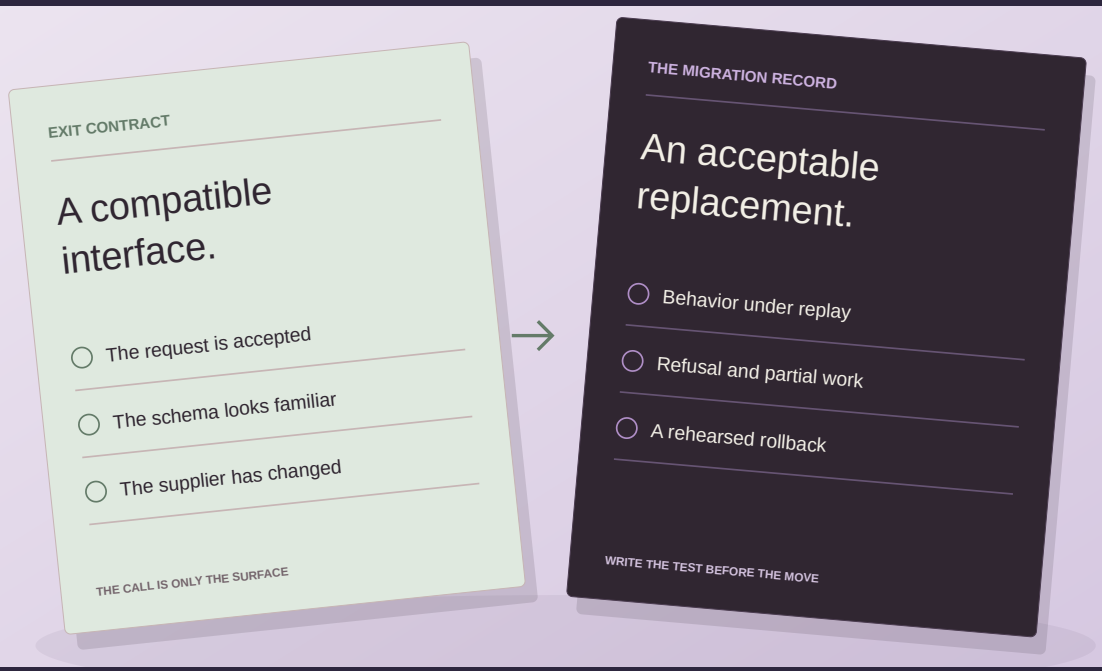
kymata labs

RESEARCH PAPER 10 / AI INFRASTRUCTURE & PORTABILITY

Before You Switch Models, Write the Exit Test.

API compatibility can move the request. Only an application contract can preserve the promise.



Kymata Labs Research
2026-11-16 · Edition 1.0

Independent analysis
kymatalabs.com

Small studio. Open worlds.

THE LONGER VIEW

Executive summary.

A model provider can be easy to replace in code and difficult to leave in practice. This paper distinguishes API compatibility from behavioral portability: the evidence that a replacement preserves the promises an application makes to its users. Inspected official SDK interfaces show why text, refusal, tool requests, interrupted output and conversation state cannot be flattened into one successful response. An executable standard-library example tests fourteen hand-authored fixture pairs through three local adapters, without calling any model. The paper then develops the separate semantic, operational, data and rollback evidence needed for a release decision. It argues for deliberate rather than accidental dependence, acknowledges the value of provider-specific capabilities, and rejects both a lowest-common-denominator architecture and a passing fixture score presented as production certification.

> Analysis of pinned official OpenAI Python, Anthropic Python, OpenAI Agents SDK and JSON Schema Test Suite repository sources captured November 16, 2026. The original local example ran fourteen hand-authored fixture pairs across three adapters: forty-two evaluations, zero external model calls and zero tool executions. Counts concern authored interpretation cases, not provider incidence or model performance. The migration framework is a proposal; live quality, latency, cost, account policies, streaming and production rollback were not evaluated.

01 / The exit is not the endpoint
02 / Same request, different promises
03 / Name the promise before naming the model
04 / Fourteen small tests, no model calls
05 / A valid shape can carry the wrong answer
06 / The conversation does not fit in the adapter
07 / The cost is a completed workflow
08 / Rehearse leaving before routing everyone
09 / The case for accepting some dependence
10 / What travels with the product
Q&A / Questions worth asking
Sources / References

© 2026 Kymata Labs LLC. All rights reserved.
Read the web edition: kymatalabs.com/research/before-you-switch-models-write-the-exit-test

The exit is not the endpoint

An AI provider can be easy to replace in code and difficult to leave in practice. The endpoint changes. The credentials change. A demonstration produces another fluent answer. Somewhere downstream, a workflow still expects the old service to mean what it meant yesterday.

That expectation is the expensive dependency. A support application might treat a returned string as a completed answer, a tool request as permission to continue, and an empty result as an invitation to retry. Those interpretations can survive unnoticed for months because one provider happens to produce the cases the application handles. Changing providers makes the assumptions visible. The new response can be perfectly legitimate under its own contract and wrong for the application consuming it.

This is a hypothetical workflow, not a reported customer incident. Its purpose is to locate the migration problem. Compatibility at the API boundary establishes that software can exchange messages. Behavioral portability requires evidence that the replacement preserves the promises made to users, within declared limits. The first is an integration property. The second is a release decision.

The distinction should change when a team writes its exit plan. An exit test written after choosing the replacement is vulnerable to becoming a description of whatever the replacement happens to do. Written earlier, it becomes a durable statement of what the product needs. It can reveal an incumbent's weakness just as readily as a candidate's. It is a contract with the application, not a loyalty test for a vendor.

This paper develops that contract from inspected SDK interfaces and a reproducible local example. It argues for portability of evidence: preserve the inputs, expected decisions, allowed exceptions and operational records needed to judge a replacement. That does not make models interchangeable. It makes their differences governable.

No live provider was evaluated for this paper. No price, outage rate or model leaderboard establishes the argument. The practical contribution is narrower and immediately usable: a fixture suite that catches structural migration errors, followed by a release method that says what those fixtures cannot establish.

Same request, different promises

The official interfaces make a casual assumption difficult to defend: a response is not necessarily a piece of text. In the captured OpenAI Python SDK, a Responses object has an output array and a generation status. Its documented status values include completed, failed, in progress, cancelled, queued and incomplete. Output messages can contain text or refusal content. The convenience property `output_text` aggregates text blocks and returns an empty string when none exist. Empty text alone therefore cannot identify why no text was produced. [1]

Function calls occupy another shape. The inspected Responses function-call type contains a tool name, a call identifier and arguments represented as a JSON string. The captured Anthropic SDK's tool-use block instead contains a name, identifier and an input dictionary. Its message has a stop reason, with

distinct values for a natural end, output limit, tool use, pause, refusal and context-window exhaustion. [2] [3]

These differences are not evidence that either interface is defective. Each makes distinctions an application needs. An adapter that extracts the first text field and discards the rest may erase those distinctions while appearing to simplify integration. A tool request is not an answered question. A refusal is not an unfinished answer. A paused turn is not a completed task.

Even within an SDK designed to accommodate multiple models, feature parity is qualified. The captured Agents SDK documentation recommends checking feature availability when mixing model shapes and warns that third-party adapters introduce another compatibility layer whose semantics vary by provider. Its model interface also carries conversation identifiers, previous-response identifiers, tool definitions, output schemas and tracing choices. The boundary is considerably wider than a prompt and a string. [5] [6]

A useful migration inventory therefore records which interface features the product actually depends on. Does it resume server-held state? Does it require a particular hosted tool? Does it parse structured output, expose partial streaming text, or preserve tool-call identifiers across turns? Unsupported capabilities require a design decision. Silently deleting them is also a decision, usually made by accident.

The source versions matter. We captured OpenAI Python SDK source declaring version 3.51.0 and Anthropic SDK source declaring 2.8.0 on November 16, 2026, with full commit hashes retained. These are source snapshots, not a claim that a package was installed or an endpoint exercised. An SDK version does not pin the provider's model weights, serving configuration or future availability. Record the requested model identifier and the identifier actually returned, where available; do not turn that record into an unsupported promise of identical future behavior.

FIGURE 01 INTERFACE CONTRACT → APPLICATION DECISION

The adapter carries a promise, not just a field.

Three boundaries separate a successful exchange from a successful replacement. Preserve the distinctions before evaluating the answer.

01

Can it be read?

Envelope

Inspect generation status, stop reason and content type. Text extraction alone loses refusal and interruption.

02

What happens next?

Workflow outcome

Preserve answer, tool proposal, refusal, incomplete and unknown outcomes. Keep call identity and validated arguments.

03

Is it acceptable?

Product promise

Evaluate task meaning, permitted data movement, operational limits and recovery. Shape compatibility does not answer these.

Analytical framework grounded in inspected interface distinctions. It is not a model comparison or a claim of feature parity.

Kymata Labs analysis of OpenAI Responses, Anthropic Messages and Agents SDK interfaces. [1] [2] [3] [5] [6]

Name the promise before naming the model

An exit contract begins at the last meaningful boundary the application owns. For the illustrative support workflow, that boundary is a result the application can route: answer, proposed ticket lookup, refusal, incomplete response, or blocked response requiring investigation. The categories should remain distinct even if the interface displays similar wording to a user.

The contract also states what is outside its scope. Our small example permits one text outcome or one lookup request. It does not support parallel tools, audio, arbitrary content blocks or resumed turns. Richer responses are blocked, not quietly flattened. That is a deliberate limitation of the example rather than a recommendation that every product accept only one block.

A production team should write the corresponding contract in the vocabulary of its own work. A drafting assistant may tolerate substantial variation in phrasing while requiring source links and human review. A ticket router may tolerate different explanations while requiring the same destination for a set of unambiguous cases. An action-taking assistant needs a separate authorization decision after a model proposes an action. “The model called the tool” does not establish that the user authorized its consequences.

Boundary	A concrete migration question	Evidence that answers it
Result	Can refusal, interruption and completion still be distinguished?	Normalized outcomes and explicit unknown cases
Tool	Are name, arguments and correlation preserved without granting authority?	Allowed-tool fixtures and separate executor policy
Meaning	Does the output meet the user's requirement?	Task-specific acceptance judgments with justified tolerances
Operation	Can the replacement meet the workflow's time and cost limits?	Measured end-to-end runs and reconciled usage
Data	Can required context move under the applicable permissions?	Data inventory, destination rules and transfer decisions

These are proposed contract boundaries, not an industry certification. Their value lies in separating questions that otherwise collapse into a single migration score. A candidate can preserve response shapes and still give worse answers. It can improve answer quality and introduce an unacceptable data destination. An average across those outcomes would obscure the decision.

Write hard constraints as gates and negotiable preferences as tradeoffs. For example, preserving a refusal state can be mandatory, while a modest increase in median latency might be acceptable if accuracy improves. Record the decision owner and rationale. Without that distinction, a team can reject a valuable replacement over harmless prose changes or accept a dangerous regression because its aggregate score looks better.

Fourteen small tests, no model calls

The accompanying Python program uses only the standard library. [8] It constructs fourteen paired fixtures with reduced envelopes resembling the documented Responses and Messages shapes. They are hand-authored examples, not captured API outputs. A shared application contract then checks how an old adapter, a deliberately incomplete candidate adapter and a contract-aware candidate adapter interpret them.

Two fixtures describe permitted behavior: a completed text answer and a single request to `lookup_ticket` with a ticket identifier. The remaining twelve exercise boundaries: explicit refusal, an output limit, an unknown state, an unknown content type, malformed tool arguments, a disallowed tool, a missing call identifier, a wrong argument type, an extra argument, empty text, multiple tools and a nonterminal response.

The deliberately incomplete adapter renames fields and returns the first block. It ignores the stopping reason and omits tool-argument checks. It is intentionally deficient, authored for this illustration; it is not code from either vendor. It preserves two of the fourteen expected outcomes. The contract-aware old

and new adapters each preserve all fourteen. Across three adapters, the program performs forty-two adapter-case evaluations, with assertions checking the expected results and agreement between the two contract-aware implementations.

The shortcut's twelve mismatches fall into six groups: five tool-contract cases, three unknown or nonterminal cases, and one each for refusal, incomplete output, empty content and multiple outcomes. These counts describe this chosen set. They do not estimate how often a production application encounters any group. Adding ten versions of one fixture would change the apparent pass rate without improving knowledge about a provider.

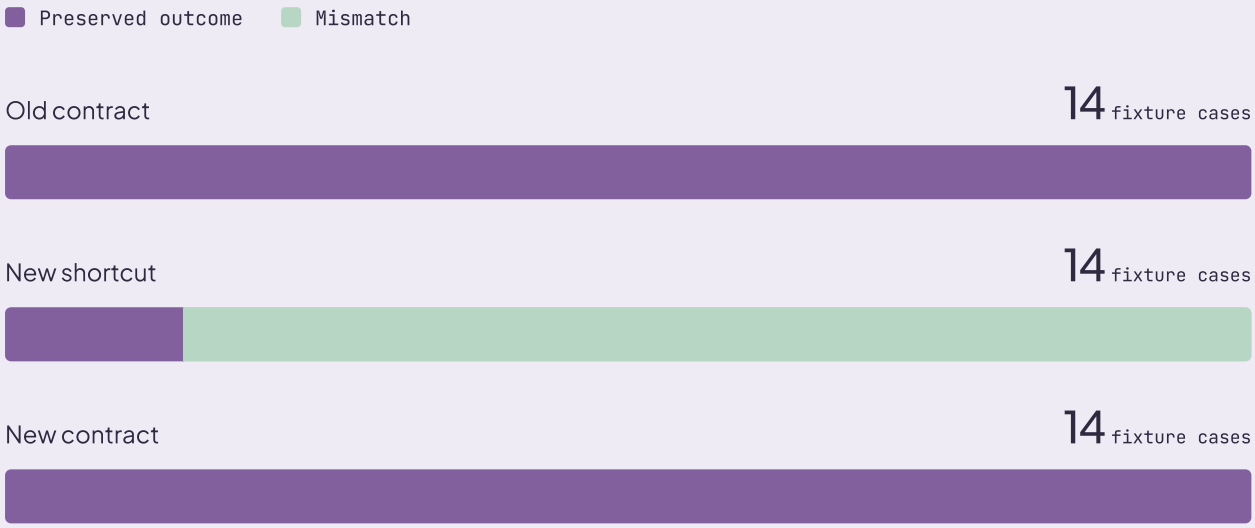
The useful result is the list of distinctions the adapter must preserve. A future stop value is blocked rather than treated as success. A tool with an extra privilege-related argument is rejected under the example's exact-field rule. A blank call identifier remains an error. The tool proposal is never executed. Passing the suite therefore means preserving a small interpretation contract, not proving that a live tool action is safe.

To reproduce the result, run `python3 example.py`. The program writes `results.json` with every expected and observed outcome, a source hash, a fixture hash and the limits of the exercise. No API key, SDK installation or network connection is required. The fixture suite is an inexpensive first gate precisely because it cannot answer the expensive questions about model behavior.

FIGURE 02 EXECUTED LOCAL ILLUSTRATION · NO MODEL CALLS

Fourteen fixtures. Twelve decisions the shortcut lost.

Each adapter interpreted the same fourteen authored cases. The deliberately incomplete shortcut drops stop-state and validation decisions.



01

14/14

Old contract adapter

Preserved every expected local outcome across permitted behavior and failure boundaries.

02

2/14

New field shortcut

Preserved only the two permitted cases. Twelve boundary outcomes mismatched.

03

14/14

New contract adapter

Preserved the same application outcomes with a different reduced response envelope.

42 adapter-case evaluations; 0 external model calls; 0 tool executions. Five tool-contract failures, three unknown/nonterminal failures, and four other mismatches in the shortcut. Authored fixture frequencies are not production rates.

Kymata Labs example.py and results.json; reproducible standard-library fixture run. [8]

A valid shape can carry the wrong answer

Schema validation is necessary for many workflows and inadequate as an evaluation of meaning. A string can name the wrong customer. A correctly shaped lookup can request the wrong ticket. A JSON object can contain every required field while inventing the facts that fill them.

The JSON Schema Test Suite provides a useful boundary around the word “valid.” Its documentation says the tests verify implementations against specified schema behavior. The inspected cases distinguish required properties, additional properties and types; they do not evaluate whether the represented

claim is true. A boolean is not an integer in the inspected integer tests, despite Python's inheritance relationship between `bool` and `int`. Strings that look numeric are also distinct from numbers. [4]

Our example does not implement or claim conformance to that suite. It checks one small application contract explicitly: a dictionary containing exactly one nonempty string field called `ticket_id`. For a real schema, use a suitable validator, declare the dialect and test the schema against the application cases. Do not substitute a successful JSON parse for validation, or validation for correctness.

The semantic exit test comes next. Select representative tasks before inspecting a candidate's outputs. Preserve the relevant context and state the reason each accepted answer is acceptable. Some tasks support exact checks; others require expert judgment. A complaint summary can differ in wording while preserving the material facts. An escalation decision may require agreement on a documented policy. Define these tolerances in advance.

Do not make the incumbent's output the unquestioned answer key. That would reward imitation of its mistakes and penalize legitimate improvements. Use the underlying task requirement as the authority, then compare both systems against it. When reviewers disagree, retain the disagreement and revise ambiguous requirements openly. A migration test should expose uncertainty rather than convert it into a polished percentage.

The conversation does not fit in the adapter

A provider change can also change where a workflow's context lives. The captured Agents model interface includes a previous-response identifier and conversation identifier. Its documentation describes prior-response references as generally specific to the OpenAI Responses API. The captured Responses type says instructions from a prior response do not carry over simply because `previous_response_id` is used. [1][6]

These details matter because a transcript is not automatically a complete execution record. The product may rely on application instructions, retrieved files, tool definitions, permissions, server-held context or the meaning of an earlier tool result. Copying visible chat turns may omit material state. Copying everything may transfer data the new destination is not allowed to receive.

An exit inventory should distinguish state the application owns from state held by a provider. For each required item, record whether it can be exported, reconstructed from an authorized source, deliberately discarded, or used only while the old workflow finishes. “Reconstructed” must remain visible: a fresh summary is a new interpretation, not the original context magically restored.

The same discipline applies to tracing. The inspected Agents interface explicitly distinguishes tracing with data, tracing without input/output data, and disabled tracing. Those are useful configuration distinctions, not proof that any particular deployment meets a privacy obligation. [6] Examine the actual trace destination and retention settings for the proposed configuration. A migration that changes inference providers while leaving full prompts in a second service has more than one data boundary.

No retention policy or contractual permission was audited for this paper. Those questions require the applicable account terms, settings and data classifications. They should be release gates where relevant, not assumptions inferred from an SDK's function names. Shadow testing is still data transfer when a second provider receives the request. A technically convenient comparison does not authorize that transfer.

For sensitive or stateful workflows, a small authorized test corpus may be the right starting point. Later evaluation can expand only within the permissions and evidence needed for the actual use case. Missing access is a limitation of the claim, not a reason to quietly substitute unrelated public prompts and announce equivalence.

The cost is a completed workflow

Migration economics are easy to simplify incorrectly. A token price is one input into the cost of delivering a result. A product also pays for retries, additional model turns, tools, retrieval, human review and failed attempts. Some of those costs fall outside an inference invoice. Some are better tracked separately than converted into a speculative monetary total.

The captured OpenAI SDK README documents configurable timeouts and automatic retries for specified errors. Its defaults are SDK behavior, not a service-level promise. The captured Anthropic message documentation explains that usage counts need not correspond one-to-one with visible request and response content. [3][7] These are reasons to measure a workflow, not grounds for asserting that one provider is cheaper.

Define the latency clock at the product boundary: for example, request accepted to a usable answer, with tool time and retries included. Record first visible output separately if it matters to the interface. Report the sample, time window and tail behavior alongside a median. A fast first token can coexist with a slow completed task; neither measure should impersonate the other.

For cost, retain billed usage and the pricing terms applicable to the run, then reconcile against billing evidence where available. Keep human review effort, setup work and rejected attempts visible. If usage data are missing, report that absence. Do not manufacture a zero or assume the same tokenizer makes nominal token counts directly comparable.

None of these operational quantities is measured by our local fixtures. Their execution time would mostly describe a tiny Python program. The article therefore reports no latency improvement, price saving or cost-per-answer estimate. The absence is intentional: successful interpretation tests are permission to investigate those questions, not evidence that they have already been answered.

Rehearse leaving before routing everyone

A release packet should connect the exact application revision, adapter revision, requested model configuration, fixture results and task evaluation to a routing decision. It should also identify the questions still open. The packet can be brief; the essential property is that a later reviewer can determine what was tested and why the decision followed.

Start with offline contract tests, then a permitted task evaluation on both the incumbent and candidate. Examine failures by task family and consequence, not only in aggregate. If ordinary requests dominate the sample, a high overall score can hide a complete failure on the rare case that justifies the product's guardrail. Conversely, a deliberately adversarial set is valuable for exposing weaknesses but cannot estimate their frequency in ordinary traffic.

Where release risk warrants it, use a bounded rollout with a declared observation window and stop conditions. Define who can stop routing, what evidence triggers the decision and what happens to work already in progress. Canary routing is a proposed operational method here; we have not deployed or measured it. Shadow evaluation requires its own data permission and must prevent duplicate effects from proposed tool actions.

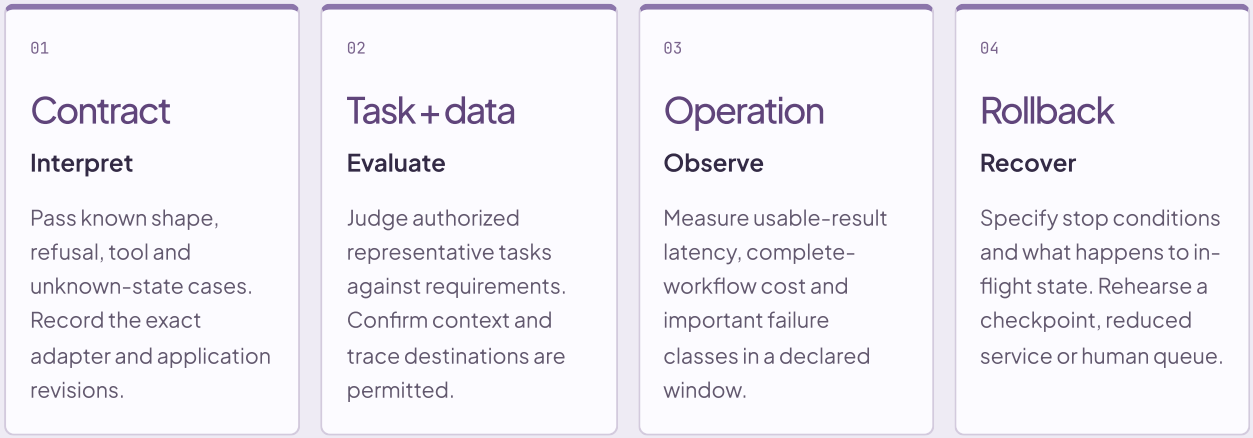
Rollback deserves a separate rehearsal. A configuration flag may redirect future requests while leaving an active workflow tied to the replacement's conversation state. Switching the next request back can fail if the old provider cannot interpret that state. Safer options may include letting an in-flight workflow finish on its original route, stopping it for human recovery, or starting again from an application-owned checkpoint. The correct choice depends on the task and side effects.

Do not assume the incumbent will remain available indefinitely. A fallback can be a reduced service, a manual queue or a clear pause, rather than another model pretending to have inherited the whole conversation. Write down what the fallback can preserve and what the user will lose. An honest degraded mode is more useful than a nominal rollback that continues under incompatible assumptions.

FIGURE 03 PROPOSED MIGRATION EVIDENCE

A release has gates. An exit needs a return path.

Move from a local contract to a bounded routing decision. Each gate answers a different question; no single score replaces the others.



The fixture exercise implements only the first gate. No live canary, quality study, billing comparison, policy audit or production rollback was performed.

Original Kymata Labs migration framework, informed by inspected interface and operational documentation. [5] [6] [7]

The case for accepting some dependence

A reasonable objection is that this program could turn a small provider switch into a migration bureaucracy. It could also flatten a product to the least capable common interface. Both outcomes would be mistakes.

Provider-specific capabilities can be worth using. Hosted tools, distinctive modalities or a workflow that performs unusually well may create value larger than the cost of eventual replacement. An exit contract should make that tradeoff visible, not forbid it. Record the dependency, its benefit and the fallback required if it disappears. Deliberate dependence is an engineering choice; unrecorded dependence is a surprise waiting for a deadline.

Nor does every application require simultaneous multi-provider operation. Maintaining a second integration creates work of its own. A small product may sensibly retain an application-owned task corpus and a documented stopping point, then build a replacement only when there is a reason. The exit test reduces uncertainty about that decision without requiring permanent duplicate infrastructure.

The scale of evidence should follow the consequence of error. A personal brainstorming tool may need a focused review of representative prompts and obvious interface failures. A tool that changes external records needs a stronger action boundary, state handling and recovery path. “We passed fourteen fixtures” is inadequate for the latter and should never be presented as a universal release standard.

There is also a positive migration case the contract should recognize. A replacement may preserve the required behavior while substantially improving quality or economics. Some wording and optional features may change, and users may prefer the result. Behavioral portability means preserving the agreed promise, not freezing every incidental characteristic of the incumbent.

What travels with the product

The evidence for this paper consists of pinned official repository sources and the executed local fixture suite. We inspected response and tool types, stop-state definitions, model integration guidance, tracing distinctions and selected JSON Schema conformance cases. Captured bytes, URLs and SHA-256 receipts are retained with a claim ledger. The examples use reduced non-streaming envelopes and do not implement the full provider protocols. They do not test model quality, streaming assembly, production resilience, billing or data-policy compliance.

The proposed exit contract is an engineering framework, not a validated intervention or a claim that migration failures are common at any measured rate. Its central proposition is nevertheless testable: before routing a workflow to a replacement, determine whether the replacement preserves the distinctions and outcomes the application has promised to respect.

The next useful step is small. Pick one real workflow. Name its acceptable outcomes, one unacceptable confusion and the state it must retain. Turn that confusion into a fixture before choosing a replacement. Then add task and operational evidence proportionate to what the workflow can do.

The most durable form of provider independence is not a universal client wrapper. It is the ability to explain what your product must continue to mean when the client changes. Carry that explanation, its tests and its limits with the product. The endpoint is replaceable. The responsibility is yours.

Questions worth asking.

Does an OpenAI-compatible API make two models interchangeable?

It can simplify how requests are sent, but it does not establish that a replacement preserves task quality, tool semantics, stop states, context or operational limits. The inspected Agents SDK documentation explicitly warns about feature differences and added adapter layers. Evaluate the exact provider path and the application behavior you intend to preserve. [5]

Should every result match the incumbent model word for word?

No. Define acceptance from the underlying task, not from the incumbent's output. A summary may vary in phrasing while preserving all material facts; a tool proposal may need an exact allowed name and validated fields. Treat refusal, incomplete output and unknown states distinctly. An exit test should recognize genuine improvements rather than reward copying existing mistakes.

What does the fourteen-case fixture result prove?

It shows that two local adapters preserve the fourteen expected outcomes in an authored set, while a deliberately incomplete shortcut preserves two. The envelopes are reduced illustrations, not captured provider responses. The result does not measure model quality, production failure frequency, streaming, cost, latency or the safety of executing tools. The Python source and detailed results are downloadable in reference 8. [8]

Should a small team run two providers all the time to avoid lock-in?

Not necessarily. Maintaining a second live integration has a cost. A bounded application-owned task corpus, explicit state inventory and documented fallback may be the proportionate starting point. Provider-specific features can justify dependence when their benefit, replacement cost and failure mode are consciously accepted. The framework does not require a lowest-common-denominator product.

Can rollback simply switch the model setting back?

That may redirect new requests, but it does not establish that an active workflow's conversation state is portable. Depending on the task, existing runs may need to finish on their original route, restart from an application-owned checkpoint or pause for human recovery. Rehearse that decision and prevent duplicate effects. A reduced service or manual queue can be a more honest fallback than pretending continuity is preserved.

SOURCES, NOT SCENERY

References.

1. **Responses object, generation states and text/refusal output types**
   OpenAI · Repository snapshot captured 2026–11–16 · Primary SDK interface source

   Inspected response.py, response_status.py, response_output_message.py and response_output_refusal.py. Source version declares 3.51.0. Supports output array, distinct generation states, text/refusal blocks, empty-string behavior of output_text and prior-response instruction handling. Does not establish model accuracy, server availability or reproducibility.

   https://github.com/openai/openai-python/blob/3dae6090cc31cc64a437fe0bcee740b6d81ed2cc/src/openai/types/responses/response.py

2. **Responses function-call representation**
   OpenAI · Repository snapshot captured 2026–11–16 · Primary SDK tool interface source

   Arguments are represented as a JSON string; name and call_id are separate fields. The fixture borrows this reduced shape but is not captured provider traffic or a complete adapter implementation.

   https://github.com/openai/openai-python/blob/3dae6090cc31cc64a437fe0bcee740b6d81ed2cc/src/openai/types/responses/response_function_tool_call.py

3. **Messages response, stop reasons and tool-use representation**
   Anthropic · Repository snapshot captured 2026–11–16 · Primary SDK interface source

   Inspected message.py, stop_reason.py, tool_use_block.py and model.py. Source version declares 2.8.0. Distinct stop reasons, tool input dictionary, identifiers and usage/visible-content distinction support the analysis. No current pricing, model ranking or service guarantee inferred.

   https://github.com/anthropics/anthropic-sdk-python/blob/5139cf2e7d07aa618a6c30b7e0c712f10ccb5ad9/src/anthropic/types/message.py

4. **JSON Schema validation test suite**
   JSON Schema organization · Repository snapshot captured 2026–11–16 · Primary conformance-test documentation and fixtures

   Inspected README plus validation/tests/type.json, required.json and additionalProperties.json. The suite tests specified schema behavior, including numeric-looking strings, booleans, required fields and extra properties. We did not run or claim conformance to this suite; our program validates one explicitly limited application contract.

   https://github.com/json-schema-org/JSON-Schema-Test-Suite/blob/38dea932b02986f496345bf007108277777630bf/validation/README.md

5. **Models: provider integration and feature differences**
   OpenAI Agents SDK contributors · Repository snapshot captured 2026–11–16 · Primary integration documentation

   Inspected provider integration, mixing model shapes, structured-output and third-party adapter sections. Documents feature differences and added adapter compatibility layers. Normative integration guidance, not independent failure-frequency measurements. The source version fallback is 0.24.0; no SDK package was installed.

   https://github.com/openai/openai-agents-python/blob/64fbde5d02b42ae5f9db668eb0184cdb3fc9d775/docs/models/index.md

6. **Model interface and tracing distinctions**
   OpenAI Agents SDK contributors · Repository snapshot captured 2026–11–16 · Primary model interface source

   Inspected model parameters, previous_response_id/conversation_id distinctions, tool call correlation and ModelTracing states. Tracing options are configuration capabilities, not a privacy-compliance determination or evidence of actual account settings.

   https://github.com/openai/openai-agents-python/blob/64fbde5d02b42ae5f9db668eb0184cdb3fc9d775/src/agents/models/interface.py

7. **OpenAI Python SDK: request IDs, retries, timeouts and versioning**
   OpenAI · Repository snapshot captured 2026–11–16 · Primary SDK operational documentation

   Inspected request tracing, configured retries, timeouts and versioning sections. Describes SDK behavior at this revision; it is not an SLA, price sheet or observed latency measurement. Source version file also captured.

   https://github.com/openai/openai-python/blob/3dae6090cc31cc64a437fe0bcee740b6d81ed2cc/README.md

8. **Model-exit fixture example and reproducible local results**
   Kymata Labs Research · 2026–11–16 · Original illustrative executable artifact

   Run python3 example.py with Python 3.10 or later; standard library only. Fourteen paired fixtures, three adapters, forty-two evaluations. Old contract and new contract each preserve 14/14 expected outcomes; deliberately incomplete shortcut preserves 2/14. No external model calls or tool executions. Results: https://www.kymatalabs.com/research/before-you-switch-models-write-the-exit-test/assets/results.json. Fixture incidence is authored and cannot estimate provider failure rates.

   https://www.kymatalabs.com/research/before-you-switch-models-write-the-exit-test/assets/example.py