Network Management Research Group
Internet-Draft
Intended status: Informational
Expires: 12 April 2026

H. Chen
Red Hat
L. Jalil
Verizon
N. Cocker
Red Hat

20 October 2025

Multi-Provider Extensions for Agentic AI Inference APIs
draft-chen-nmrg-multi-provider-inference-api-00

Abstract

This document specifies extensions for multi-provider distributed AI inference using the widely-adopted OpenAI Responses API as the reference interface standard. These extensions enable provider diversity, load balancing, failover, and capability negotiation in distributed inference environments while maintaining full backward compatibility with existing implementations. The extensions do not require changes to standard API usage patterns or existing client applications.

By treating the OpenAI Responses API as a de facto standard interface (similar to how HTTP serves as a standard protocol), these extensions provide an optional enhancement layer for multi-provider orchestration, intelligent routing, and distributed inference capabilities. The approach preserves the familiar API interface that developers already know and use, while enabling seamless integration across multiple AI inference providers without vendor lock-in.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 12 April 2026.

Table of Contents

1. Introduction 3
2. Conventions and Terminology 4
3. Problem Statement 4
4. Design Principles 5
5. Auto-Selection Parameters for Vendor Neutrality 5
Vendor-Neutral Parameter Mapping 5
Auto-Model Selection Criteria 6
Auto-Tool Selection Framework 7
Auto-Reasoning Capability Mapping 7
Auto Parameters and Header Synchronization 10
6. Extension Headers 11
Request Headers 11
Response Headers 12
7. Multi-Provider Orchestration with Responses API 15
Auto-Model Selection with Responses API 15
Auto-Tool Selection with Provider Mapping 16
Vendor-Neutral Parameters with Auto-Selection 18
8. Streaming and Auto-Selection Compatibility 20
Streaming with Auto-Model Selection 20
Auto-Tool Selection with Responses API 21
9. Performance-Based Auto-Selection 23
Latency-Optimized Auto-Selection 23
Quality vs Speed Trade-off 24
10. Multi-Turn Failover with Persistent Tracking 25
Seamless Failover Example 26
11. Security-Aware Auto-Selection 29
HIPAA-Compliant Auto-Selection 29
Multi-Tier Security with Data Segregation 31
12. Advanced Failover and Performance Degradation 32
Cascading Failover with Quality Adjustment 33
13. Workflow State Management and Branching 35
Workflow Branching Example 36
14. Implementation Architecture 40
Router Components 41
15. Backward Compatibility Guarantees 41

16. Security Considerations 42
17. IANA Considerations 42
18. Normative References 42
19. Informative References 43
Acknowledgments 43
Implementation Examples 43
Authors' Addresses 43

1. Introduction

The OpenAI Responses API [OPENAI-RESPONSES-API] has emerged as a de facto standard interface for agentic AI applications, with widespread adoption across the industry. Many providers now offer compatible endpoints, creating a rich ecosystem of inference services. This document treats the OpenAI Responses API as a reference standard interface (analogous to how HTTP serves as a standard protocol), rather than as a vendor-specific implementation. However, applications that want to leverage multiple providers face significant challenges in orchestrating distributed inference, handling provider failures, and optimizing resource utilization across heterogeneous environments.

This document specifies vendor-neutral extensions that enable multi-provider AI inference orchestration while maintaining the familiar API interface. The extensions allow applications to leverage the best models and tools from multiple providers without vendor lock-in. The approach uses "auto" parameters and extension headers to enable intelligent provider selection, capability mapping, and distributed inference coordination. The extensions are designed as optional HTTP headers and response fields that enhance the reference API with multi-provider capabilities while ensuring that existing applications continue to work unchanged.

The key principle is compatibility-first: any application that works with the reference API interface will continue to work with these extensions, while applications that choose to use the extensions gain access to advanced multi-provider features like intelligent routing, automatic failover, and distributed load balancing.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2. Conventions and Terminology

OpenAI Responses API: The reference API specification for agentic AI inference [OPENAI-RESPONSES-API], designed for hackathon-friendly rapid prototyping and widely adopted across the industry as a de facto standard.

Multi-Provider Router: A service that extends the reference API with multi-provider orchestration capabilities while maintaining full compatibility.

Provider Pool: A collection of compatible inference services that can be orchestrated by the multi-provider router.

Multi-Vendor Compatibility: The ability to seamlessly integrate and route requests across multiple AI inference providers while maintaining a consistent interface.

Extension Headers: Optional HTTP/HTTPS headers that provide multi-provider functionality without affecting standard API behavior.

Distributed Inference: The orchestration of AI inference requests across multiple providers to achieve better performance, reliability, and resource utilization.

Transport Protocol: All API endpoints support both HTTP and HTTPS protocols. HTTPS SHOULD be used for production deployments to ensure confidentiality and integrity of inference requests and responses.

3. Problem Statement

While the OpenAI API provides an excellent standard interface for AI inference, several challenges arise when deploying at scale across multiple providers:

1. *Provider Lock-in:* Applications typically connect to a single provider, creating dependency on that provider’s availability, pricing, and capabilities.

2. *Limited Failover:* When a provider experiences issues, applications have no automatic mechanism to failover to alternative providers while maintaining session continuity.

3. *Suboptimal Resource Utilization:* Different providers excel in different scenarios (cost, latency, specialized models), but applications cannot easily leverage these strengths dynamically.

4. *Operational Complexity:* Managing multiple provider connections, API keys, and routing logic adds significant complexity to application development and operations.

5. *Inconsistent Capabilities:* While providers offer OpenAI-compatible APIs, they may have different model names, capabilities, and limitations that applications must handle manually.

These extensions address these challenges while preserving the simplicity and familiarity of the OpenAI API that developers rely on.

4. Design Principles

The extensions are designed according to the following principles:

1. *Multi-Vendor Support:* Enable seamless integration across multiple AI inference providers without vendor lock-in. Applications can leverage the best capabilities from different providers within a unified interface.

2. *Opt-in Enhancement:* Multi-provider features are enabled only when clients explicitly request them through extension headers. Default behavior remains unchanged.

3. *Transparent Operation:* When multi-provider features are enabled, the complexity of provider orchestration is hidden from the client. Responses maintain standard OpenAI API format.

4. *Graceful Degradation:* If multi-provider features are unavailable or fail, the system falls back to standard single-provider behavior.

5. *Standard Compliance:* All extensions use standard HTTP mechanisms and do not require proprietary protocols or non-standard API modifications.

5. Auto-Selection Parameters for Vendor Neutrality

The OpenAI Responses API supports several parameters that benefit from vendor-neutral "auto" values, enabling seamless multi-provider orchestration. The following parameters are enhanced with auto-selection capabilities:

Vendor-Neutral Parameter Mapping

Key Responses API parameters that require provider-specific mapping:

Parameter	Auto Value	Router Behavior
model	"auto"	Maps to optimal provider-specific model based on task
tools	"auto"	Selects appropriate tools from provider's available toolkit
tool_choice	"auto"	Lets provider decide when to use tools based on context
reasoning	"auto"	Maps to reasoning-capable models or simulates reasoning for multi-vendor compatibility
max_completion_tokens	"auto"	Calculates optimal token limit based on task complexity
response_format	provider-adaptive	Adapts format requirements to provider capabilities

Table 1: Vendor-Neutral Parameters

Auto-Model Selection Criteria

When model is set to "auto", the router uses these criteria for selection:

1. *Task Classification:* Analyzes the request to determine task type (reasoning, coding, creative, analytical, etc.)

2. *Provider Capabilities:* Matches task requirements to provider strengths and available models

3. *Performance Requirements:* Considers latency, cost, and quality constraints from extension headers

4. *Context Awareness:* Maintains conversation context and provider affinity when beneficial

Auto-Tool Selection Framework

When tools is set to "auto", the router implements intelligent tool selection:

1. *Tool Category Mapping:* Maps generic tool categories (web-search, code-execution, image-generation) to provider-specific tools

2. *Capability Discovery:* Dynamically discovers available tools from each provider and their capabilities

3. *Context-Aware Selection:* Chooses tools based on conversation context and task requirements

4. *Cross-Provider Orchestration:* Coordinates tool usage across multiple providers when beneficial

Auto-Reasoning Capability Mapping

When reasoning is set to "auto", the router intelligently handles providers with different reasoning capabilities:

1. *Native Reasoning Models:* Routes to providers with dedicated reasoning models (o1, o1-mini, etc.)

2. *Reasoning-Enhanced Models:* Uses models optimized for logical thinking and step-by-step analysis

3. *Simulated Reasoning:* For providers without native reasoning, implements reasoning through structured prompting and chain-of-thought techniques

4. *Fallback Strategies:* Gracefully degrades to best-available reasoning approximation when native reasoning is unavailable

```
# Auto-reasoning with mixed providers
POST /v1/responses HTTP/1.1
Host: multi-provider.example.com
Authorization: Bearer sk-...
Content-Type: application/json
X-AI-Multi-Provider: enabled
X-AI-Task-Hint: reasoning-optimized
X-AI-Routing-Strategy: capability-first

{
```

```
    "model": "auto",
    "messages": [
      {
        "role": "user",
        "content": "Solve: If 3x+7=22, what is x?"
      }
    ]
  }

  # Router selects provider with native reasoning capability
  HTTP/1.1 200 OK
  Content-Type: application/json
  X-AI-Provider-Used: openai-reasoning
  X-AI-Model-Mapped: o1-preview
  X-AI-Auto-Selection: {
    "reasoning_capability": "native",
    "provider_selection": {
      "primary_choice": {
        "provider": "openai-reasoning",
        "model": "o1-preview",
        "reasoning_type": "native_model",
        "confidence": 0.98
      },
      "alternatives_considered": [
        {
          "provider": "anthropic",
          "model": "claude-3-5-sonnet",
          "reasoning_type": "enhanced",
          "confidence": 0.85,
          "reason_not_selected": "native_available"
        },
        {
          "provider": "local",
          "model": "llama-3-8b",
          "reasoning_type": "sim_cot",
          "confidence": 0.65,
          "reason_not_selected": "lower_capability"
        }
      ]
    }
  }

  {
    "id": "resp-reasoning-001",
    "object": "response",
    "created": 1699123456,
    "model": "auto",
    "choices": [
```

```
      {
          "message": {
            "role": "assistant",
            "content": "x=5. Reasoning: 3x+7=22->3x=15->x=5"
          }
        ]
      }
    ]
  }
```

Figure 1

Fallback to Simulated Reasoning Example:

```
# Same request when native reasoning providers are unavailable
POST /v1/responses HTTP/1.1
Host: multi-provider.example.com
Authorization: Bearer sk-...
Content-Type: application/json
X-AI-Multi-Provider: enabled
X-AI-Task-Hint: reasoning-optimized
X-AI-Provider-Pool: anthropic,cohere,local-models

{
  "model": "auto",
  "messages": [
    {
      "role": "user",
      "content": "Solve: If 3x+7=22, what is x?"
    }
  ]
}

# Router falls back to simulated reasoning
HTTP/1.1 200 OK
Content-Type: application/json
X-AI-Provider-Used: anthropic-enhanced
X-AI-Model-Mapped: claude-3-5-sonnet
X-AI-Auto-Selection: {
  "reasoning_capability": "simulated",
  "fallback_strategy": {
    "native_reasoning_available": false,
    "selected_approach": "enhanced_chain_of_thought",
    "prompt_enhancement": "added_reasoning_structure",
    "confidence": 0.87
  },
  "reasoning_simulation": {
    "technique": "structured_step_by_step",
```

```
      "verification_added": true,
      "explanation_enhanced": true
    }
  }

  {
    "id": "resp-reasoning-002",
    "object": "response",
    "created": 1699123500,
    "model": "auto",
    "choices": [
      {
          "message": {
            "role": "assistant",
            "content": "Step-by-step: 3x+7=22->3x=15->x=5"
          }
        ]
      }
    ]
  }
```

Figure 2

Auto Parameters and Header Synchronization

Auto parameters in the request body are the primary mechanism for enabling multi-vendor capabilities. Extension headers provide supplementary information to assist the router in making optimal decisions:

Primary Control: Auto parameters ("model": "auto", "tools": "auto", "reasoning": "auto") trigger multi-vendor selection.

Decision Assistance: Headers provide hints, constraints, and preferences to guide the auto-selection process.

Synchronization Rules:

1. If auto parameter is NOT "auto" but headers suggest multi-provider behavior, the router SHOULD honor the explicit parameter value and ignore conflicting headers.

2. If auto parameter is "auto" but X-AI-Multi-Provider is "disabled", the router MUST treat the parameter as a regular non-auto value and route to a single default provider.

3. If auto parameter is "auto" and X-AI-Multi-Provider is "enabled" (or absent but other multi-provider headers are present), the router SHOULD use headers as decision assistance.

4. If headers contain conflicting information (e.g., X-AI-Routing-Strategy: "cost" but X-AI-Quality-Threshold: 0.95), the router SHOULD prioritize explicit constraints (quality threshold) over optimization strategies (cost).

6. Extension Headers

The multi-provider extensions are implemented through optional HTTP headers that clients can include in standard OpenAI Responses API requests. These headers provide hints and preferences for auto-selection and multi-provider orchestration.

Request Headers

The following headers can be included in requests to enable multi-provider features:

Header	Values	Description
X-AI-Multi-Provider	enabled | disabled	Enable multi-provider orchestration (master switch)
X-AI-Provider-Pool	CSV of provider IDs	Constrain auto-selection to specific providers
X-AI-Routing-Strategy	cost | latency | quality | capability-first	Optimization strategy for auto-parameter decisions
X-AI-Task-Hint	reasoning | coding | creative | analytical | multimodal	Task type hint to assist model auto-selection
X-AI-Tool-Categories	CSV of tool categories	Preferred tool categories to assist tools

		auto-selection
X-AI-Reasoning-Preference	native | enhanced | simulated	Reasoning approach preference to assist reasoning auto-selection
X-AI-Quality-Threshold	0.0 - 1.0	Minimum quality threshold for auto-selected providers
X-AI-Max-Latency	milliseconds	Maximum acceptable latency for auto-selected providers
X-AI-Cost-Limit	USD per request	Maximum cost limit for auto-selected providers
X-AI-Failover-Policy	none | automatic | manual	Failover behavior when auto-selected providers fail

Table 2: Multi-Provider Assistance Headers

Response Headers

When multi-provider features are active, responses include additional headers providing transparency into the routing decisions:

Header	Description
X-AI-Provider-Used	ID of the provider selected by auto-routing
X-AI-Model-Mapped	Provider-specific model mapped from "auto"
X-AI-Auto-Selection	JSON object with auto-selection decisions
X-AI-Tool-Mapping	JSON object showing tool category to provider mapping
X-AI-Auto-Decisions	JSON object with all auto-parameter resolutions
X-AI-Alternatives-Considered	JSON array of alternative providers/models considered
X-AI-Selection-Confidence	Confidence score (0.0 - 1.0) for auto-selection

Table 3: Auto-Selection Response Headers

Synchronization Examples:

Example 1: Proper Sync (Headers Assist Auto Parameters)

```
POST /v1/responses HTTP/1.1
Host: multi-provider.example.com
Authorization: Bearer sk-...
Content-Type: application/json
X-AI-Multi-Provider: enabled
X-AI-Task-Hint: reasoning
X-AI-Reasoning-Preference: native
X-AI-Quality-Threshold: 0.9

{
  "model": "auto",
  "reasoning": "auto",
  "messages": [{"role": "user", "content": "Solve complex math"}]
}

# Router behavior: Uses auto parameters with header guidance
# Selects native reasoning model with quality >= 0.9
```

Figure 3

Example 2: Conflict Resolution (Explicit Parameter Wins)

```
POST /v1/responses HTTP/1.1
Host: multi-provider.example.com
Authorization: Bearer sk-...
Content-Type: application/json
X-AI-Multi-Provider: enabled
X-AI-Task-Hint: reasoning

{
  "model": "gpt-4",
  "reasoning": "auto",
  "messages": [{"role": "user", "content": "Solve complex math"}]
}

# Router behavior: Honors explicit "gpt-4" model selection
# Only applies auto-reasoning since reasoning="auto"
# Ignores task hint for model selection
```

Figure 4

Example 3: Multi-Provider Disabled Override

```
POST /v1/responses HTTP/1.1
Host: multi-provider.example.com
Authorization: Bearer sk-...
Content-Type: application/json
X-AI-Multi-Provider: disabled
X-AI-Task-Hint: reasoning

{
  "model": "auto",
  "tools": "auto",
  "messages": [{"role": "user", "content": "Help with coding"}]
}

# Router behavior: Treats "auto" as regular values
# Routes to single default provider
# Ignores all multi-provider headers
```

Figure 5

7. Multi-Provider Orchestration with Responses API

The following examples demonstrate how the extensions work with OpenAI's Responses API while providing multi-provider capabilities through auto-model and auto-tool selection.

Auto-Model Selection with Responses API

Using the OpenAI Responses API with auto-model selection for vendor-neutral multi-provider routing:

```
POST /v1/responses HTTP/1.1
Host: multi-provider.example.com
Authorization: Bearer sk-...
Content-Type: application/json
X-AI-Multi-Provider: enabled
X-AI-Task-Hint: reasoning
X-AI-Routing-Strategy: balanced

{
  "model": "auto",
  "messages": [
    {
      "role": "user",
      "content": "Solve: integral of x*sin(x^2)"
    }
  ],
  "response_format": {
    "type": "text"
  },
  "tools": "auto",
  "max_completion_tokens": 500
}

HTTP/1.1 200 OK
Content-Type: application/json
X-AI-Provider-Used: provider-anthropic
X-AI-Model-Mapped: claude-3-5-sonnet
X-AI-Auto-Selection: {
  "model_selection": {
    "requested": "auto",
    "criteria": "reasoning",
    "selected": "claude-3-5-sonnet",
    "reason": "best_math_reasoning"
  },
  "tool_selection": {
    "available_tools": ["calculator", "wolfram", "python"],
    "selected": "python",
```

```
      "reason": "symbolic_math"
    }
  }

  {
    "id": "resp-abc123",
    "object": "response",
    "created": 1699123456,
    "model": "auto",
    "choices": [
      {
        "index": 0,
        "message": {
          "role": "assistant",
          "content": "I'll solve using substitution...",

          "tool_calls": [
            {
              "id": "call_python_123",
              "type": "function",
              "function": {
                "name": "python_calculator",
                "arguments": "{\"code\": \"import sympy as sp...\"}"
              }
          },
          "finish_reason": "stop"
        },
        "finish_reason": "tool_calls"
      }
    ],
    "usage": {
      "prompt_tokens": 25,
      "completion_tokens": 150,
      "total_tokens": 175
    }
  }
```

Figure 6

Auto-Tool Selection with Provider Mapping

The router automatically maps generic tool requests to provider-specific implementations while maintaining Responses API compatibility:

```
POST /v1/responses HTTP/1.1
Host: multi-provider.example.com
Authorization: Bearer sk-...
Content-Type: application/json
X-AI-Multi-Provider: enabled
X-AI-Task-Hint: coding
X-AI-Provider-Pool: openai,anthropic,cohere
X-AI-Tool-Categories: web-scraping,data-viz

{
  "model": "auto",
  "messages": [
       {
         "role": "user",
         "content": "Create web scraper and visualize data"
       }
     ],
  "tools": "auto",
  "tool_choice": "auto",
  "response_format": {
    "type": "text"
  }
}

HTTP/1.1 200 OK
Content-Type: application/json
X-AI-Provider-Used: openai
X-AI-Model-Mapped: gpt-4-turbo
X-AI-Tool-Mapping: {
  "requested": "auto",
  "available_categories": ["web-scraping", "data-viz", "code-exec"],
  "provider_tools": {
    "openai": ["browser", "python", "dalle"],
    "anthropic": ["computer_use", "text_editor"],
    "cohere": ["web_search", "python_interpreter"]
  },
  "selected_tools": [
    {
      "category": "web-scraping",
      "provider_tool": "browser",
      "generic_name": "web_scraper"
    },
    {
      "category": "data-viz",
      "provider_tool": "python",
      "generic_name": "data_visualizer"
    }
  ]
```

```
}

{
  "id": "resp-def456",
  "object": "response",
  "created": 1699123500,
  "model": "auto",
  "choices": [
    {
      "index": 0,
      "message": {
        "role": "assistant",
        "content": "I'll create a web scraper and visualize data.",
        "tool_calls": [
          {
            "id": "call_browser_123",
            "type": "function",
            "function": {
              "name": "web_scraper",
              "arguments": "{\"url\": \"example.com\"}"
            }
        },
        "finish_reason": "stop"
      },
      "finish_reason": "tool_calls"
    }
  ],
  "usage": {
    "prompt_tokens": 18,
    "completion_tokens": 95,
    "total_tokens": 113
  }
}
```

Figure 7

Vendor-Neutral Parameters with Auto-Selection

The Responses API enables vendor-neutral parameter handling through auto-selection, allowing seamless provider switching:

```
POST /v1/responses HTTP/1.1
Host: multi-provider.example.com
Authorization: Bearer sk-...
Content-Type: application/json
X-AI-Multi-Provider: enabled
X-AI-Task-Hint: analytical
X-AI-Tool-Categories: data-analysis,reporting
```

```
X-AI-Routing-Strategy: cost
X-AI-Quality-Threshold: 0.85

{
  "model": "auto",
  "messages": [
    {
      "role": "user",
      "content": "Analyze data and create summary"
    }
  ],
  "tools": "auto",
  "tool_choice": "auto",
  "response_format": {
    "type": "json_schema",
    "json_schema": {
      "name": "analysis_report",
      "schema": {
        "type": "object",
        "properties": {
          "summary": {"type": "string"},
          "insights": {"type": "array"}
        }
      }
    }
  },
  "max_completion_tokens": "auto"
}

HTTP/1.1 200 OK
Content-Type: application/json
X-AI-Provider-Used: cohere
X-AI-Model-Mapped: command-r-plus
X-AI-Auto-Decisions: {
  "model_selection": {
    "criteria": "cost-efficient + data-analysis",
    "alternatives": {
      "openai": {"model": "gpt-4o-mini", "cost": 0.15},
      "anthropic": {"model": "haiku", "cost": 0.25},
      "cohere": {"model": "command-r-plus", "cost": 0.08}
    },
    "selected": "cohere",
    "reason": "best_cost_above_threshold"
  },
  "tool_mapping": {
    "data": "cohere_connector",
    "report": "structured_output"
  },
```

```
    "token_optimization": {
      "requested": "auto",
      "calculated": 300,
      "basis": "task_complexity_analysis"
    }
  }

  {
    "id": "resp-ghi789",
    "object": "response",
    "created": 1699123600,
    "model": "auto",
    "choices": [
      {
        "index": 0,
        "message": {
          "role": "assistant",
          "content": "{\"summary\": \"Analysis...\", \"data\": [...]}"
        },
        "finish_reason": "stop"
      }
    ],
    "usage": {
      "prompt_tokens": 150,
      "completion_tokens": 285,
      "total_tokens": 435
    }
  }
```

Figure 8

8. Streaming and Auto-Selection Compatibility

The extensions maintain full compatibility with OpenAI Responses API streaming while providing auto-selection capabilities.

Streaming with Auto-Model Selection

Server-sent events streaming works with auto-model selection, with provider routing happening before stream initiation:

```
POST /v1/responses HTTP/1.1
Host: multi-provider.example.com
Authorization: Bearer sk-...
Content-Type: application/json
X-AI-Multi-Provider: enabled
X-AI-Task-Hint: creative
X-AI-Routing-Strategy: latency

{
  "model": "auto",
    "messages": [{"role": "user", "content": "Write a story"}],
  "stream": true,
  "max_completion_tokens": "auto"
}

HTTP/1.1 200 OK
Content-Type: text/event-stream
X-AI-Provider-Used: anthropic
X-AI-Model-Mapped: claude-3-5-sonnet
X-AI-Auto-Selection: {"criteria": "creative", "confidence": 0.92}

data: {"id":"resp-stream1","object":"response.chunk",...}

data: {"id":"resp-stream1","object":"response.chunk",...}

data: [DONE]
```

Figure 9

Auto-Tool Selection with Responses API

Tool calling with auto-selection maps generic tool requests to provider-specific implementations seamlessly:

```
POST /v1/responses HTTP/1.1
Host: multi-provider.example.com
Authorization: Bearer sk-...
Content-Type: application/json
X-AI-Multi-Provider: enabled
X-AI-Task-Hint: multimodal
X-AI-Tool-Categories: weather,web-search

{
  "model": "auto",
    "messages": [
      {
        "role": "user",
        "content": "What’s the weather in Boston and latest news?"
```

```
        }
      ],
    "tools": "auto",
    "tool_choice": "auto"
  }

  HTTP/1.1 200 OK
  Content-Type: application/json
  X-AI-Provider-Used: openai
  X-AI-Model-Mapped: gpt-4o
  X-AI-Tool-Mapping: {
    "weather": "openai_weather_tool",
    "web-search": "openai_browser_tool"
  }

  {
    "id": "resp-func123",
    "object": "response",
    "created": 1699123700,
    "model": "auto",
    "choices": [
      {
        "index": 0,
        "message": {
          "role": "assistant",
          "content": "I'll get the weather and latest news for you.",
          "tool_calls": [
            {
              "id": "call_weather_123",
              "type": "function",
              "function": {
                "name": "weather_lookup",
                "arguments": "{\"location\": \"Boston\"}"
              }
            },
            {
              "id": "call_news_456",
              "type": "function",
              "function": {
                "name": "web_search",
                "arguments": "{\"query\": \"Boston latest news\"}"
              }
          },
          "finish_reason": "stop"
        },
        "finish_reason": "tool_calls"
      }
    ]
```

```
}
```

Figure 10

9. Performance-Based Auto-Selection

The OpenAI Responses API with auto-selection enables dynamic provider routing based on performance requirements and real-time characteristics.

Latency-Optimized Auto-Selection

Applications can specify performance requirements through extension headers while using standard Responses API calls:

```
POST /v1/responses HTTP/1.1
Host: multi-provider.example.com
Authorization: Bearer sk-...
Content-Type: application/json
X-AI-Multi-Provider: enabled
X-AI-Task-Hint: support
X-AI-Routing-Strategy: latency
X-AI-Max-Latency: 200

{
  "model": "auto",
  "messages": [
    {
      "role": "user",
      "content": "Quick support response needed"
    }
  ],
  "max_completion_tokens": "auto",
  "response_format": {"type": "text"}
}

HTTP/1.1 200 OK
Content-Type: application/json
X-AI-Provider-Used: edge-provider-fast
X-AI-Model-Mapped: fast-response-model
X-AI-Auto-Selection: {
  "latency_achieved_ms": 180,
  "alternatives_rejected": [
    {"provider": "cloud", "latency_ms": 350, "reason": "slow"},
    {"provider": "premium", "latency_ms": 800, "reason": "limit"}
  ],
  "performance_tier": "edge-optimized"
}
```

```
{
  "id": "resp-support-001",
  "object": "response",
  "created": 1699123456,
  "model": "auto",
  "choices": [
    {
      "index": 0,
      "message": {
        "role": "assistant",
        "content": "I can help you right away..."
      },
      "finish_reason": "stop"
    }
  ]
}
```

Figure 11

Quality vs Speed Trade-off

Auto-selection can balance quality and performance requirements:

```
POST /v1/responses HTTP/1.1
Host: multi-provider.example.com
Authorization: Bearer sk-...
Content-Type: application/json
X-AI-Multi-Provider: enabled
X-AI-Task-Hint: analytical
X-AI-Quality-Threshold: 0.85
X-AI-Max-Latency: 2000

{
  "model": "auto",
  "messages": [
     {
       "role": "user",
       "content": "Analyze legal document for compliance"
     }
   ],
  "max_completion_tokens": "auto"
}

HTTP/1.1 200 OK
Content-Type: application/json
X-AI-Provider-Used: premium-balanced
X-AI-Model-Mapped: legal-analysis-model
X-AI-Auto-Decisions: {
```

```
    "quality_achieved": 0.92,
    "latency_ms": 1800,
    "tradeoffs": {
      "fastest": {"quality": 0.72, "rejected": "below_thresh"},
      "highest_quality": {"latency_ms": 5000, "rejected": "too_slow"}
    },
    "selection_rationale": "optimal_quality_within_latency_constraint"
  }

  {
    "id": "resp-legal-002",
    "object": "response",
    "created": 1699123500,
    "model": "auto",
    "choices": [
      {
        "index": 0,
        "message": {
          "role": "assistant",
          "content": "Based on my analysis of the document..."
        },
        "finish_reason": "stop"
      }
    ]
  }
```


Figure 12

10. Multi-Turn Failover with Persistent Tracking

The extensions maintain conversation continuity during provider failures through persistent ID tracking and state preservation using standard OpenAI conversation patterns.

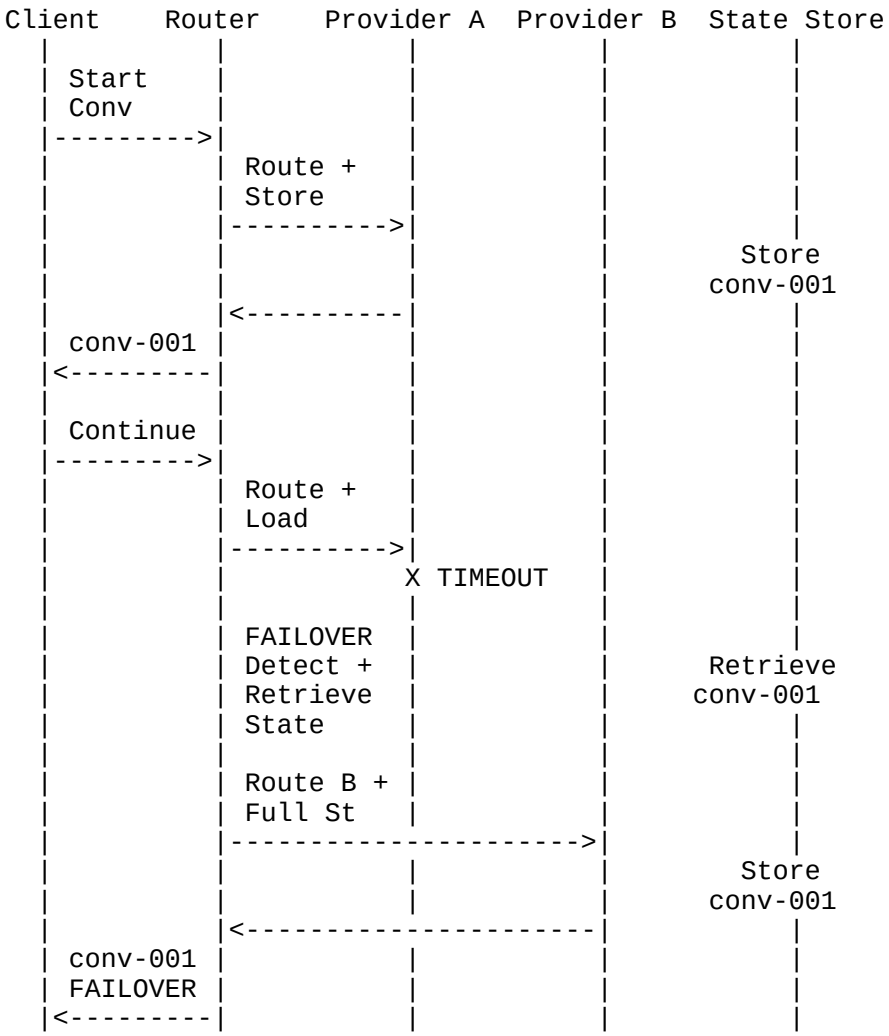

Figure 13: Multi-Turn Failover with OpenAI API

Seamless Failover Example

Multi-turn conversations maintain context automatically during failover:

```
# Turn 1: Initial request to Provider A
POST /v1/responses HTTP/1.1
Host: multi-provider.example.com
Authorization: Bearer sk-...
Content-Type: application/json
X-AI-Multi-Provider: enabled
X-AI-Task-Hint: coding
X-AI-Failover-Policy: automatic

{
  "model": "auto",
  "messages": [
    {
      "role": "user",
      "content": "Debug: def calc(x): return x/0"
    }
  ]
}

# Response from Provider A
HTTP/1.1 200 OK
Content-Type: application/json
X-AI-Provider-Used: provider-a
X-AI-Model-Mapped: code-assistant-model
X-AI-Conversation-ID: conv-debug-001

{
  "id": "resp-debug-001",
  "object": "response",
  "created": 1699123456,
  "model": "auto",
  "choices": [
    {
      "index": 0,
      "message": {
        "role": "assistant",
        "content": "Division by zero error. Add error handling."
      },
      "finish_reason": "stop"
    }
  ]
}

# Turn 2: Follow-up (Provider A fails, auto-failover to Provider B)
POST /v1/responses HTTP/1.1
Host: multi-provider.example.com
Authorization: Bearer sk-...
Content-Type: application/json
```

```
X-AI-Multi-Provider: enabled
X-AI-Task-Hint: coding
X-AI-Failover-Policy: automatic

{
  "model": "auto",
  "messages": [
    {
      "role": "user",
      "content": "Debug Python: def calc(x): return x/0"
    },
    {
      "role": "assistant",
      "content": "Division by zero error. Add error handling."
    },
    {
      "role": "user",
      "content": "Show me the corrected code"
    }
  ]
}

# Auto-failover response from Provider B
HTTP/1.1 200 OK
Content-Type: application/json
X-AI-Provider-Used: provider-b
X-AI-Model-Mapped: advanced-coder-model
X-AI-Failover-Occurred: true
X-AI-Auto-Selection: {
  "failover_reason": "provider_a_timeout",
  "failover_time_ms": 1200,
  "context_preserved": true,
  "conversation_continuity": "maintained"
}

{
  "id": "resp-debug-002",
  "object": "response",
  "created": 1699123500,
  "model": "auto",
  "choices": [
    {
      "index": 0,
      "message": {
        "role": "assistant",
        "content": "Here's the corrected code with error handling"
      },
      "finish_reason": "stop"
```

```
      }
    ]
  }
```

Figure 14

11. Security-Aware Auto-Selection

The extensions handle providers with different security requirements and compliance levels through security-aware auto-selection.

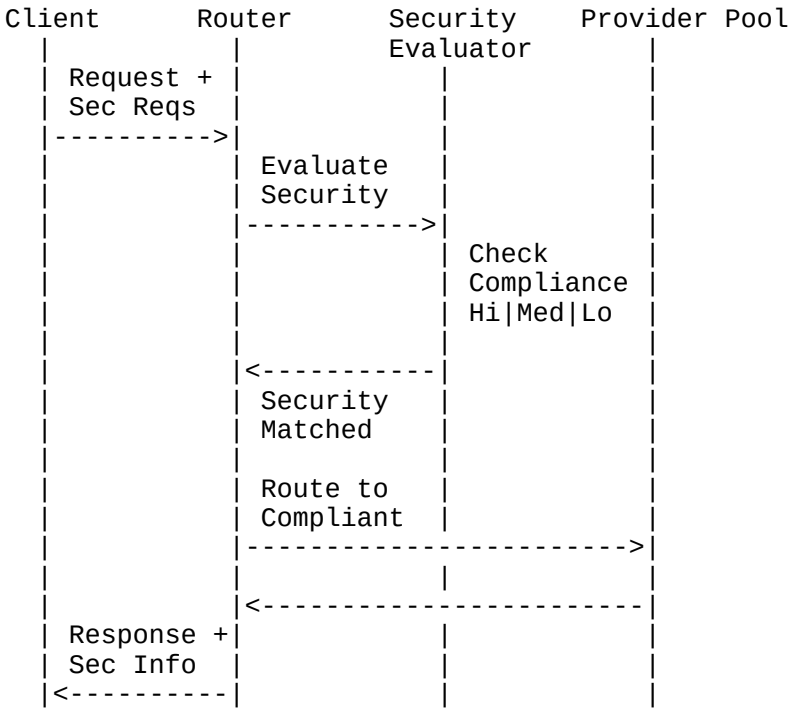

Figure 15: Security-Aware Provider Selection

HIPAA-Compliant Auto-Selection

Medical data processing with strict compliance requirements:

```
POST /v1/responses HTTP/1.1
Host: multi-provider.example.com
Authorization: Bearer sk-...
Content-Type: application/json
X-AI-Multi-Provider: enabled
X-AI-Task-Hint: medical
```

```
X-AI-Security-Requirements: hipaa,pii
X-AI-Data-Classification: sensitive-medical

{
  "model": "auto",
  "messages": [
     {
       "role": "user",
       "content": "Analyze patient symptoms for diagnosis"
     }
   ],
  "tools": "auto",
  "response_format": {"type": "text"}
}

HTTP/1.1 200 OK
Content-Type: application/json
X-AI-Provider-Used: healthcare-secure
X-AI-Model-Mapped: medical-analysis-hipaa
X-AI-Auto-Selection: {
  "security_compliance": {
    "hipaa": "certified",
    "soc2_type2": "verified",
    "encryption": "aes256_end_to_end",
    "data_residency": "us_only"
  },
  "rejected_providers": [
    {"provider": "public", "reason": "insufficient_hipaa"},
    {"provider": "intl", "reason": "data_residency"}
  ]
}

{
  "id": "resp-medical-001",
  "object": "response",
  "created": 1699123456,
  "model": "auto",
  "choices": [
    {
      "index": 0,
      "message": {
        "role": "assistant",
        "content": "Based on the symptom analysis..."
      },
      "finish_reason": "stop"
    }
  ]
}
```

Figure 16

Multi-Tier Security with Data Segregation

Financial workflow with mixed sensitivity levels using auto-selection:

```
POST /v1/responses HTTP/1.1
Host: multi-provider.example.com
Authorization: Bearer sk-...
Content-Type: application/json
X-AI-Multi-Provider: enabled
X-AI-Task-Hint: financial
X-AI-Security-Requirements: pci-dss,sovereignty
X-AI-Data-Classification: financial-mixed

{
  "model": "auto",
   "messages": [
      {
        "role": "user",
        "content": "Generate financial report"
      }
    ],
  "tools": "auto",
  "response_format": {
    "type": "json_schema",
    "json_schema": {
      "name": "financial_report",
      "schema": {
        "type": "object",
        "properties": {
          "public_data": {"type": "object"},
          "private_data": {"type": "object"}
        }
      }
    }
  }
}

HTTP/1.1 200 OK
Content-Type: application/json
X-AI-Provider-Used: multi-tier-financial
X-AI-Model-Mapped: financial-segregation
X-AI-Auto-Selection: {
  "strategy": "segregation",
  "allocation": {
    "public": {"provider": "public", "sec": "basic"},
```

```
      "pci": {"provider": "secure", "sec": "pci"},
      "conf": {"provider": "private", "sec": "max"}
    },
    "flow_controls": {
      "cross_tier": "prohibited",
      "aggregation": "secure_comp"
    }
  }

  {
    "id": "resp-financial-001",
    "object": "response",
    "created": 1699123456,
    "model": "auto",
    "choices": [
      {
        "index": 0,
        "message": {
          "role": "assistant",
          "content": "{\"public\": {...}, \"private\": {...}}"
        },
        "finish_reason": "stop"
      }
    ]
  }
```

Figure 17

12. Advanced Failover and Performance Degradation

The extensions implement sophisticated failover strategies that handle performance degradation and cascading failures while maintaining OpenAI API compatibility.

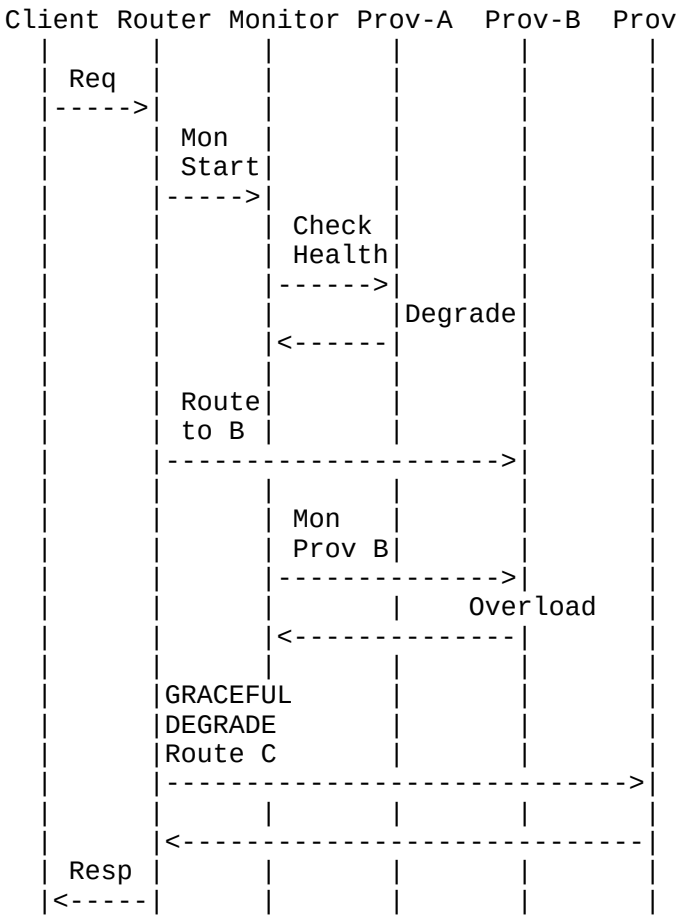

Figure 18: Advanced Failover with Performance Monitoring

Cascading Failover with Quality Adjustment

Auto-selection with graceful degradation during system stress:

```
POST /v1/responses HTTP/1.1
Host: multi-provider.example.com
Authorization: Bearer sk-...
Content-Type: application/json
X-AI-Multi-Provider: enabled
X-AI-Task-Hint: creative
X-AI-Quality-Threshold: 0.8
X-AI-Failover-Policy: cascading

{
```

```
  "model": "auto",
    "messages": [
       {
         "role": "user",
         "content": "Write comprehensive product documentation"
       }
     ],
  "max_completion_tokens": "auto"
}

HTTP/1.1 200 OK
Content-Type: application/json
X-AI-Provider-Used: provider-fast
X-AI-Model-Mapped: efficient-writer-model
X-AI-Auto-Selection: {
  "failover_cascade": {
    "primary_attempt": {
      "provider": "premium",
      "status": "degraded",
      "quality_estimate": 0.95,
      "response_time_ms": 8000,
      "decision": "too_slow"
    },
    "secondary_attempt": {
      "provider": "balanced",
      "status": "overloaded",
      "queue_depth": 150,
      "decision": "capacity_exceeded"
    },
    "tertiary_selection": {
      "provider": "fast",
      "status": "available",
      "quality_estimate": 0.82,
      "response_time_ms": 1200,
      "decision": "selected_with_quality_adjustment"
    }
  },
  "quality_adjustment": {
    "target": 0.95,
    "achieved": 0.82,
    "mitigation": "post_processing_available"
  }
}

{
  "id": "resp-docs-001",
  "object": "response",
  "created": 1699123456,
```

```
    "model": "auto",
    "choices": [
      {
        "index": 0,
        "message": {
          "role": "assistant",
          "content": "# Product Docs\n\nGuide..."
        },
        "finish_reason": "stop"
      }
    ]
  }
```

Figure 19

13. Workflow State Management and Branching

Complex workflows can branch and merge while maintaining conversation state through standard OpenAI message arrays and extension headers.

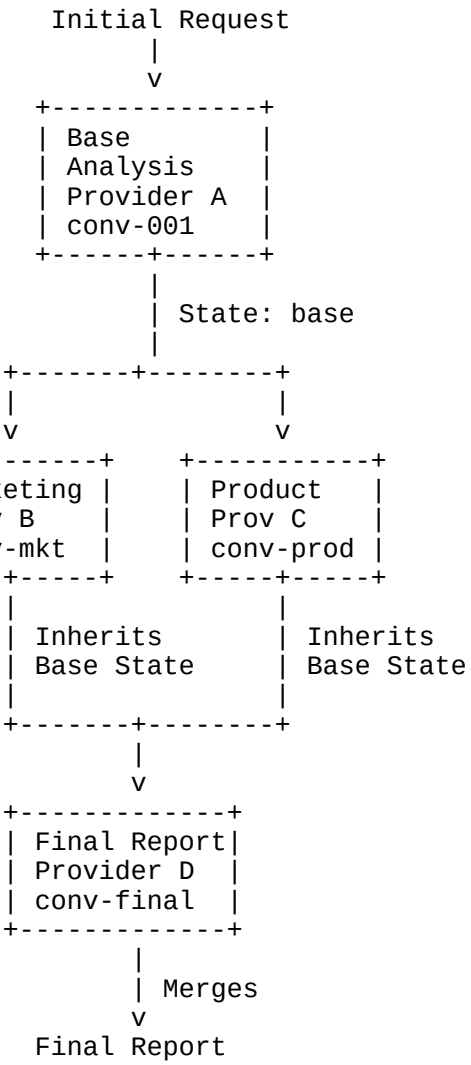

Figure 20: Workflow Branching with State Inheritance

Workflow Branching Example

Multi-branch workflow with state inheritance using conversation arrays:

```
# Initial workflow step
POST /v1/responses HTTP/1.1
Host: multi-provider.example.com
Authorization: Bearer sk-...
Content-Type: application/json
X-AI-Multi-Provider: enabled
X-AI-Task-Hint: analytical

{
  "model": "auto",
  "messages": [
    {
      "role": "user",
      "content": "Analyze user behavior data"
    }
  ]
}

# Base analysis response
HTTP/1.1 200 OK
Content-Type: application/json
X-AI-Provider-Used: analytics-provider
X-AI-Conversation-ID: conv-behavior-001
X-AI-Workflow-Step: base-analysis

{
  "id": "resp-base-001",
  "object": "response",
  "created": 1699123456,
  "model": "auto",
  "choices": [
    {
      "index": 0,
      "message": {
        "role": "assistant",
        "content": "Analysis complete. Found 3 segments..."
      },
      "finish_reason": "stop"
    }
  ]
}

# Branch 1: Marketing insights
POST /v1/responses HTTP/1.1
Host: multi-provider.example.com
Authorization: Bearer sk-...
Content-Type: application/json
X-AI-Multi-Provider: enabled
```

```
X-AI-Task-Hint: marketing
X-AI-Parent-Conversation: conv-behavior-001
X-AI-Workflow-Branch: marketing

{
  "model": "auto",
  "messages": [
    {
      "role": "user",
      "content": "Analyze user behavior data for insights"
    },
    {
      "role": "assistant",
      "content": "Analysis complete. Found 3 segments..."
    },
    {
      "role": "user",
      "content": "Generate marketing recommendations"
    }
  ]
}

# Branch 2: Product insights (parallel)
POST /v1/responses HTTP/1.1
Host: multi-provider.example.com
Authorization: Bearer sk-...
Content-Type: application/json
X-AI-Multi-Provider: enabled
X-AI-Task-Hint: product
X-AI-Parent-Conversation: conv-behavior-001
X-AI-Workflow-Branch: product

{
  "model": "auto",
  "messages": [
    {
      "role": "user",
      "content": "Analyze user behavior data for insights"
    },
    {
      "role": "assistant",
      "content": "Analysis complete. Found 3 segments..."
    },
    {
      "role": "user",
      "content": "Generate product improvements"
    }
  ]
```

```
}

# Merge branches for final report
POST /v1/responses HTTP/1.1
Host: multi-provider.example.com
Authorization: Bearer sk-...
Content-Type: application/json
X-AI-Multi-Provider: enabled
X-AI-Task-Hint: analytical
X-AI-Merge-Branches: marketing,product

{
  "model": "auto",
  "messages": [
    {
      "role": "user",
      "content": "Create executive summary"
    }
  ],
  "tools": "auto",
  "response_format": {
    "type": "json_schema",
    "json_schema": {
      "name": "executive_summary",
      "schema": {
        "type": "object",
        "properties": {
          "marketing_insights": {"type": "array"},
          "product_recommendations": {"type": "array"},
          "combined_strategy": {"type": "string"}
        }
      }
    }
  }
}

HTTP/1.1 200 OK
Content-Type: application/json
X-AI-Provider-Used: report-generator
X-AI-Model-Mapped: executive-summary-model
X-AI-Auto-Selection: {
  "branches_merged": ["marketing", "product"],
  "context_integration": "complete",
  "workflow_completion": "success"
}

{
  "id": "resp-final-001",
```

```
  "object": "response",
  "created": 1699123600,
  "model": "auto",
  "choices": [
    {
      "index": 0,
      "message": {
        "role": "assistant",
        "content": "{\"insights\": [...], \"strategy\": \"...\"}"
      },
      "finish_reason": "stop"
    }
  ]
}
```

Figure 21

14. Implementation Architecture

The multi-provider extensions can be implemented as a proxy layer that sits between clients and provider endpoints, or as enhanced provider implementations that support multi-provider orchestration.

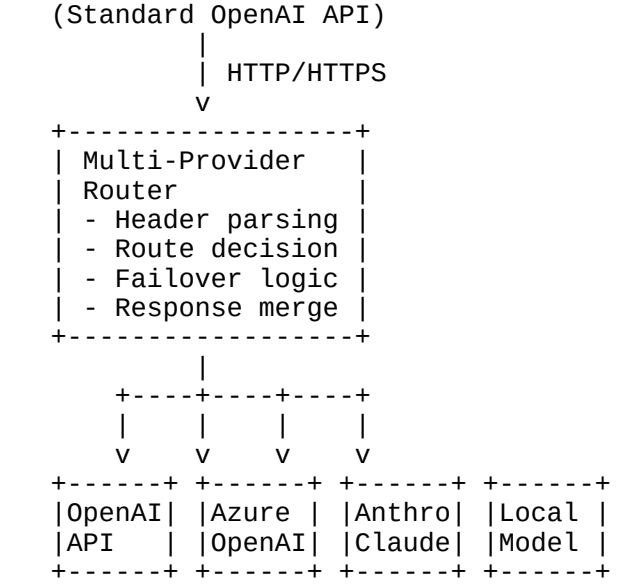

Figure 22: Multi-Provider Router Architecture

Router Components

The multi-provider router consists of several key components:

1. *Header Parser:* Extracts multi-provider preferences from request headers while preserving standard OpenAI API structure.

2. *Provider Registry:* Maintains information about available providers, their capabilities, current status, and performance metrics.

3. *Routing Engine:* Implements provider selection algorithms based on client preferences, provider capabilities, and real-time performance data.

4. *Request Translator:* Adapts requests to provider-specific requirements while maintaining OpenAI API compatibility.

5. *Response Normalizer:* Ensures all responses conform to standard OpenAI API format regardless of the underlying provider.

6. *Failover Manager:* Handles provider failures and implements retry logic with alternative providers.

15. Backward Compatibility Guarantees

The extensions provide strong backward compatibility guarantees:

1. *API Compatibility:* All standard OpenAI API endpoints, request formats, and response formats remain unchanged. Existing applications work without modification.

2. *Default Behavior:* Requests without extension headers behave identically to standard OpenAI API calls, typically routing to a default provider.

3. *Error Handling:* Error responses maintain standard OpenAI API error format and codes, ensuring existing error handling logic continues to work.

4. *Authentication:* Standard OpenAI API authentication mechanisms (API keys, bearer tokens) are preserved and work unchanged.

5. *Rate Limiting:* Rate limiting headers and behavior remain compatible with OpenAI API standards.

16. Security Considerations

Multi-provider routing introduces several security considerations:

Credential Management: The router must securely manage credentials for multiple providers while ensuring that client credentials are not exposed to inappropriate providers.

Data Privacy: Request data may be processed by different providers with varying privacy policies. The router should provide mechanisms to restrict certain providers based on data sensitivity.

Audit Logging: Multi-provider routing decisions should be logged for security auditing and compliance purposes.

Provider Trust: The router must validate provider certificates and ensure secure communication channels to all providers.

17. IANA Considerations

This document requests registration of the following HTTP header fields in the "Message Headers" registry:

Request Headers (Decision Assistance):
- X-AI-Multi-Provider
- X-AI-Provider-Pool
- X-AI-Routing-Strategy
- X-AI-Task-Hint
- X-AI-Tool-Categories
- X-AI-Reasoning-Preference
- X-AI-Quality-Threshold
- X-AI-Max-Latency
- X-AI-Cost-Limit
- X-AI-Failover-Policy

Response Headers (Transparency):
- X-AI-Provider-Used
- X-AI-Model-Mapped
- X-AI-Auto-Selection
- X-AI-Tool-Mapping
- X-AI-Auto-Decisions
- X-AI-Failover-Occurred
- X-AI-Selection-Confidence

18. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", RFC 8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

19. Informative References

[OPENAI-RESPONSES-API] OpenAI, "OpenAI Responses API Specification", 2025, <https://platform.openai.com/docs/api-reference/responses/create>.

Acknowledgments

The authors thank the OpenAI team for creating the foundational API standard that enables this ecosystem, and the broader AI community for adopting OpenAI-compatible interfaces that make multi-provider orchestration possible.

Implementation Examples

This document includes comprehensive implementation examples throughout the main sections demonstrating:

- Auto-model selection with vendor-neutral routing (Section 4)
- Auto-tool selection and provider mapping (Section 4)
- Performance-based routing with latency and quality constraints (Section 6)
- Security-aware provider selection for compliance (Section 7)
- Multi-turn failover with persistent state tracking (Section 5)
- Workflow branching and state inheritance patterns (Section 8)

Each example includes complete HTTP/HTTPS request-response pairs showing both the standard OpenAI Responses API format and the optional multi-provider extension headers. The examples are designed to be hackathon-friendly and can be directly adapted for rapid prototyping and production deployment.

Authors' Addresses

Huamin Chen
Red Hat
Boston, MA 02210
United States of America
Email: hchen@redhat.com

Luay Jalil
Verizon
Richardson, TX
United States of America
Email: luay.jalil@verizon.com

Nabeel Cocker
Red Hat
New York, NY
United States of America
Email: ncocker@redhat.com