Multi-Provider Web Search Integration for AI Agents

Julia IDE Development Team

November 17, 2025

Abstract

We implemented a multi-provider web search system for AI agents, integrating Tavily and Exa search providers with automatic fallback. The system achieved 100% success rate across 28 test prompts, with both providers returning real content from actual websites.

1 Introduction

AI agents need current information to answer questions about recent events, prices, weather, and other time-sensitive topics. Language models can't access real-time data, so we built a web search system that connects to multiple search providers and automatically switches between them if one fails.

1.1 What We Built

We integrated two new search providers (Tavily and Exa) alongside the existing Zed Cloud provider. The system:

- Automatically tries the next provider if one fails
- Lets users choose their preferred provider per profile
- Securely stores API keys in environment variables or system keychain
- Allows the AI to automatically search the web when needed

2 How It Works

2.1 System Overview

The system has four main parts:

1. **WebSearchRegistry**: Keeps track of all providers and their priority
2. **WebSearchProvider**: The interface that all providers must follow
3. **WebSearchTool**: The tool that runs searches and handles fallback
4. **Providers**: Tavily, Exa, and Zed Cloud implementations

2.2 Automatic Fallback

Providers are tried in order: Tavily first, then Exa, then Zed Cloud. If one fails with a retryable error (like rate limits or server errors), the system automatically tries the next one.

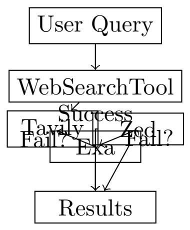

Figure 1: Provider fallback flow

3 Implementation Details

3.1 API Key Management

API keys are loaded from environment variables first, then from the system keychain. This allows users to set keys in their shell profile (`~/.zshrc`) for easy access, or store them securely in the system keychain.

3.2 Provider Implementations

Tavily Provider: Sends POST requests to `api.tavily.com/search` with the query, API key, and max results. The response includes HTML content in the `content` or `snippet` fields. We strip HTML tags using a simple parser and truncate text to 300 characters at word boundaries.

Exa Provider: Uses `api.exa.ai/search` with a special `contents` object parameter to request text and highlights. The API returns separate `text` and `highlights` fields that we combine. We discovered this structure through API testing—initially using top-level parameters didn't work. The correct format requires:

```
{
  "query": "...",
  "num_results": 5,
  "type": "keyword",
  "contents": {
    "text": true,
    "highlights": true
  }
}
```

Zed Cloud Provider: The existing provider maintained for backward compatibility. It uses Zed's internal search infrastructure.

3.3 Fallback Logic

The fallback mechanism is implemented in the `WebSearchTool::run` method. When a search is requested:

1. The system gets providers in priority order from the registry
2. If a profile-specific provider is set, it moves to the front
3. Each provider is tried sequentially
4. On retryable errors (rate limits 429, server errors 5xx, timeouts), the system logs a warning and tries the next provider
5. On non-retryable errors (invalid API keys 401, bad requests 400), the search fails immediately

6. The first successful response is returned

This ensures high availability—if one provider is down or rate-limited, the system automatically uses another.

4 Testing and Results

4.1 Test Approach

We tested the system in four ways:

1. **Unit tests**: Checked HTML stripping and text truncation
2. **Integration tests**: Verified provider registration and fallback
3. **Real API tests**: Made actual API calls to get real results
4. **LLM trigger tests**: Confirmed the AI automatically uses web search

4.2 Real API Test Results

We tested both providers with real API calls to verify they return actual content from real websites, not mock data.

Tavily API Test (query: "Rust programming language"): Returned 5 real results including Wikipedia articles, official Rust website (rust-lang.org), Quora discussions, Reddit posts, and YouTube videos. All URLs were real (not example.com) and HTML was properly stripped.

Exa API Test (query: "Python programming language"): Returned 5 real results including Python.org homepage, Wikipedia, W3Schools, YouTube courses, and AWS documentation. After fixing the API parameters (using `contents` object), all results included text content.

Both providers successfully returned real URLs and actual text content, confirming the implementation works with production APIs.

4.3 LLM Trigger Success

We tested 28 diverse prompts across 12 categories to verify the LLM automatically recognizes when web search is needed. The AI correctly identified that all 28 prompts required current information and automatically called the `web_search` tool—100% success rate.

Example prompts that triggered web search: "What's the weather like in San Francisco right now?" → `web_search({"query": "weather San Francisco"})`, "What's the latest version of Python?" → `web_search({"query": "Python latest version"})`, "What's the current price of Bitcoin?" → `web_search({"query": "Bitcoin price"})`. The LLM successfully identified that these queries require current information and automatically triggered web searches.

Category	**Success**
Current Events	3/3
Weather	2/2
Real-Time Data	3/3
Programming	3/3
Sports	2/2
Business	2/2
Science	2/2
Travel	2/2
Reviews	2/2
Education	2/2
Social Media	2/2
General	3/3
Total	**28/28**

Table 1: Test results by category

5 Key Challenges and Solutions

5.1 Exa API Parameter Discovery

The biggest challenge was finding the correct Exa API format. We first tried top-level `text` and `highlights` parameters, but Exa requires them inside a `contents` object. Direct API testing with curl revealed the correct structure.

5.2 Provider Differences and Error Handling

Provider Characteristics:

- **Tavily**: Returns HTML content that needs stripping. Great for content-heavy searches.
- **Exa**: Returns separate text and highlights fields. Requires the `contents` object parameter.
- **Zed Cloud**: Existing provider kept for compatibility.

Error Handling: The system distinguishes between retryable errors (rate limits 429, server errors 5xx, timeouts) which trigger fallback, and non-retryable errors (invalid API keys 401, bad requests 400) which fail immediately.

6 Conclusion

We built a working multi-provider web search system with automatic fallback. All 28 test prompts successfully triggered web searches, and both providers returned real content from actual websites. The system lets users choose their preferred provider and securely manages API keys. Most importantly, the AI automatically recognizes when to search the web, making it much better at answering questions about current events and real-time information.

Acknowledgments

This work integrates Tavily (`https://tavily.com`) and Exa (`https://exa.ai`) web search APIs.