OpenAI Provider Integration Guide

Overview

The OpenAI integration provides comprehensive support for OpenAI's language models including GPT-4o, GPT-4o-mini, GPT-4 Turbo, and GPT-3.5 Turbo. This integration includes advanced features like function calling, vision processing, and streaming responses.

Table of Contents

Setup and Configuration

Environment Variables

```
# Required
OPENAI_API_KEY=sk-proj-your-api-key-here

# Optional Configuration
OPENAI_ORG_ID=org-your-organization-id
OPENAI_MAX_RETRIES=3
OPENAI_TIMEOUT=30000
OPENAI_BASE_URL=https://api.openai.com/v1
OPENAI_DEFAULT_MODEL=gpt-4o-mini
```

Configuration Object

```
const openaiConfig = {
  apiKey: process.env.OPENAI_API_KEY,
  organization: process.env.OPENAI_ORG_ID,
  maxRetries: 3,
  timeout: 30000,
  baseURL: 'https://api.openai.com/v1',
  defaultHeaders: {
    'User-Agent': 'AI-Employee-Platform/1.0'
  }
};
```

Initialization

```
import { OpenAIAdvancedIntegration } from '@ai-platform/ai-routing-service';

const openaiIntegration = new OpenAIAdvancedIntegration(openaiConfig);

// Initialize with health check
await openaiIntegration.initialize();
```

Model Support

Available Models

Model	Description	Max Tokens	Cost (per 1K tokens)	Best Use Cases
`gpt-4o`	Most capable model	128K	Input: $0.005, Output: $0.015	Complex reasoning, analysis
`gpt-4o-mini`	Fast, cost-effective	128K	Input: $0.00015, Output: $0.0006	General tasks, high volume
`gpt-4-turbo`	Balanced performance	128K	Input: $0.01, Output: $0.03	Creative tasks, coding
`gpt-3.5-turbo`	Fast and affordable	16K	Input: $0.0005, Output: $0.0015	Simple tasks, chat

Model Selection Guidelines

```
const modelSelection = {
  // For simple Q&A and basic tasks
  simple: 'gpt-4o-mini',

  // For complex analysis and reasoning
  complex: 'gpt-4o',

  // For creative content generation
  creative: 'gpt-4-turbo',

  // For high-volume, cost-sensitive applications
  volume: 'gpt-3.5-turbo',

  // For vision tasks
  vision: 'gpt-4o'
};
```

Basic Usage

Simple Text Generation

```
const response = await openaiIntegration.generateResponse({
  model: 'gpt-4o-mini',
  messages: [
    {
      role: 'user',
      content: 'Explain quantum computing in simple terms'
    }
  ],
  maxTokens: 200,
  temperature: 0.7
});

console.log(response.content);
```

Chat Conversations

```
const conversation = [
  { role: 'system', content: 'You are a helpful AI assistant.' },
  { role: 'user', content: 'What is machine learning?' },
  { role: 'assistant', content: 'Machine learning is a subset of AI...' },
  { role: 'user', content: 'Can you give me an example?' }
];

const response = await openaiIntegration.generateResponse({
  model: 'gpt-4o-mini',
  messages: conversation,
  maxTokens: 300
});
```

Response Structure

```typescript
interface OpenAIResponse {
  success: boolean;
  content: string;
  usage: {
    promptTokens: number;
    completionTokens: number;
    totalTokens: number;
  };
  metadata: {
    model: string;
    provider: 'openai';
    latency: number;
    cost: number;
    finishReason: 'stop' | 'length' | 'function_call';
  };
  functionCalls?: FunctionCall[];
  error?: string;
}
```

Advanced Features

Temperature and Creativity Control

```typescript
// More deterministic (temperature = 0.1)
const factualResponse = await openaiIntegration.generateResponse({
  model: 'gpt-4o-mini',
  messages: [{ role: 'user', content: 'What is 2+2?' }],
  temperature: 0.1
});

// More creative (temperature = 0.9)
const creativeResponse = await openaiIntegration.generateResponse({
  model: 'gpt-4-turbo',
  messages: [{ role: 'user', content: 'Write a creative story' }],
  temperature: 0.9
});
```

Top-P (Nucleus) Sampling

```typescript
const response = await openaiIntegration.generateResponse({
  model: 'gpt-4o',
  messages: [{ role: 'user', content: 'Generate diverse ideas for...' }],
  temperature: 0.8,
  topP: 0.9 // Consider tokens with top 90% probability mass
});
```

Frequency and Presence Penalties

```
const response = await openaiIntegration.generateResponse({
  model: 'gpt-4o-mini',
  messages: [{ role: 'user', content: 'Write about innovation' }],
  frequencyPenalty: 0.5, // Reduce repetition of frequent tokens
  presencePenalty: 0.3,  // Reduce repetition of any tokens
  maxTokens: 500
});
```

Function Calling

Basic Function Calling

```
const weatherFunction = {
  name: 'get_weather',
  description: 'Get current weather for a location',
  parameters: {
    type: 'object',
    properties: {
      location: {
        type: 'string',
        description: 'City name, e.g. San Francisco, CA'
      },
      unit: {
        type: 'string',
        enum: ['celsius', 'fahrenheit'],
        description: 'Temperature unit'
      }
    },
    required: ['location']
  }
};

const response = await openaiIntegration.generateResponse({
  model: 'gpt-4o',
  messages: [
    { role: 'user', content: 'What\'s the weather in Tokyo?' }
  ],
  functions: [weatherFunction],
  functionCall: 'auto'
});

// Check if function was called
if (response.functionCalls && response.functionCalls.length > 0) {
  const functionCall = response.functionCalls[0];
  console.log('Function called:', functionCall.name);
  console.log('Arguments:', functionCall.arguments);

  // Execute function and continue conversation
  const weatherData = await executeWeatherFunction(functionCall.arguments);

  const followUp = await openaiIntegration.generateResponse({
    model: 'gpt-4o',
    messages: [
      { role: 'user', content: 'What\'s the weather in Tokyo?' },
      {
        role: 'assistant',
        content: null,
        functionCall: {
          name: functionCall.name,
          arguments: JSON.stringify(functionCall.arguments)
        }
      },
      {
        role: 'function',
        name: functionCall.name,
        content: JSON.stringify(weatherData)
      }
    ]
  });
}
```

Multiple Function Support

```
const functions = [
  {
    name: 'get_time',
    description: 'Get current time',
    parameters: { type: 'object', properties: {} }
  },
  {
    name: 'calculate',
    description: 'Perform mathematical calculations',
    parameters: {
      type: 'object',
      properties: {
        expression: { type: 'string', description: 'Math expression' }
      },
      required: ['expression']
    }
  }
];

const response = await openaiIntegration.generateResponse({
  model: 'gpt-4o',
  messages: [
    { role: 'user', content: 'What time is it and what is 15 * 23?' }
  ],
  functions: functions,
  functionCall: 'auto'
});
```

Parallel Function Calling

```
// GPT-4o supports parallel function calls
const response = await openaiIntegration.generateResponse({
  model: 'gpt-4o',
  messages: [
    {
      role: 'user',
      content: 'Get weather for New York and London, then calculate travel time'
    }
  ],
  functions: [weatherFunction, travelTimeFunction],
  functionCall: 'auto',
  parallelFunctionCalling: true
});

// Handle multiple function calls
if (response.functionCalls) {
  const results = await Promise.all(
    response.functionCalls.map(async (call) => {
      return await executeFunctionCall(call);
    })
  );
}
```

Vision Processing

Basic Image Analysis

```
const response = await openaiIntegration.generateResponse({
  model: 'gpt-4o',
  messages: [
    {
      role: 'user',
      content: [
        {
          type: 'text',
          text: 'What do you see in this image?'
        },
        {
          type: 'image_url',
          image_url: {
            url: 'https://example.com/image.jpg'
          }
        }
      ]
    }
  ],
  maxTokens: 300
});
```

Base64 Image Processing

```
const base64Image = 'iVBORw0KGgoAAAANSUhEUgAA...'; // Base64 encoded image

const response = await openaiIntegration.generateResponse({
  model: 'gpt-4o',
  messages: [
    {
      role: 'user',
      content: [
        { type: 'text', text: 'Describe this image in detail' },
        {
          type: 'image_url',
          image_url: {
            url: `${base64Image}`,
            detail: 'high' // 'low', 'high', or 'auto'
          }
        }
      ]
    }
  ]
});
```

Multiple Image Analysis

```
const images = [
  'https://i.ytimg.com/vi/sI_2b5VZBjM/maxresdefault.jpg',
  'https://upload.wikimedia.org/wikipedia/commons/thumb/d/da/Face_or_vase_ata_01.svg/
250px-Face_or_vase_ata_01.svg.png'
];

const response = await openaiIntegration.generateResponse({
  model: 'gpt-4o',
  messages: [
    {
      role: 'user',
      content: [
        { type: 'text', text: 'Compare these two images' },
        ...images.map(url => ({
          type: 'image_url',
          image_url: { url }
        }))
      ]
    }
  ],
  maxTokens: 500
});
```

Vision with Function Calling

```
const analyzeImageFunction = {
  name: 'analyze_image_objects',
  description: 'Detect and count objects in an image',
  parameters: {
    type: 'object',
    properties: {
      objects: {
        type: 'array',
        items: {
          type: 'object',
          properties: {
            name: { type: 'string' },
            count: { type: 'number' },
            confidence: { type: 'number' }
          }
        }
      }
    }
  }
};

const response = await openaiIntegration.generateResponse({
  model: 'gpt-4o',
  messages: [
    {
      role: 'user',
      content: [
        { type: 'text', text: 'Analyze objects in this image' },
        { type: 'image_url', image_url: { url: imageUrl } }
      ]
    }
  ],
  functions: [analyzeImageFunction],
  functionCall: 'auto'
});
```

Streaming Responses

Basic Streaming

```
const stream = await openaiIntegration.streamResponse({
  model: 'gpt-4o-mini',
  messages: [
    { role: 'user', content: 'Tell me a long story about AI' }
  ],
  stream: true,
  maxTokens: 1000
});

for await (const chunk of stream) {
  process.stdout.write(chunk);
}
```

Streaming with Metadata

```
const stream = await openaiIntegration.streamResponse({
  model: 'gpt-4o',
  messages: [{ role: 'user', content: 'Explain machine learning' }],
  stream: true
});

let fullContent = '';
let tokenCount = 0;

for await (const chunk of stream) {
  if (chunk.type === 'content') {
    fullContent += chunk.content;
    process.stdout.write(chunk.content);
  } else if (chunk.type === 'metadata') {
    tokenCount = chunk.usage?.totalTokens || 0;
    console.log('\nTokens used:', tokenCount);
  }
}
```

Streaming with Function Calls

```
const stream = await openaiIntegration.streamResponse({
  model: 'gpt-4o',
  messages: [
    { role: 'user', content: 'What time is it and tell me a joke?' }
  ],
  functions: [getTimeFunction],
  functionCall: 'auto',
  stream: true
});

for await (const chunk of stream) {
  if (chunk.type === 'content') {
    console.log('Content:', chunk.content);
  } else if (chunk.type === 'function_call') {
    console.log('Function called:', chunk.functionCall);

    // Execute function
    const result = await executeFunction(chunk.functionCall);

    // Continue with function result
    // Note: This requires additional API call to continue conversation
  }
}
```

Error Handling

Common Error Types

```
try {
  const response = await openaiIntegration.generateResponse(request);
} catch (error) {
  switch (error.type) {
    case 'insufficient_quota':
      console.error('Quota exceeded:', error.message);
      break;

    case 'invalid_request_error':
      console.error('Invalid request:', error.message);
      break;

    case 'rate_limit_exceeded':
      console.error('Rate limit hit:', error.message);
      // Implement exponential backoff
      break;

    case 'api_error':
      console.error('API error:', error.message);
      break;

    case 'network_error':
      console.error('Network issue:', error.message);
      // Implement retry logic
      break;

    default:
      console.error('Unknown error:', error);
  }
}
```

Retry Logic with Exponential Backoff

```
async function generateWithRetry(request: AIRequest, maxRetries = 3): Promise<AIRe-
sponse> {
  for (let attempt = 1; attempt <= maxRetries; attempt++) {
    try {
      return await openaiIntegration.generateResponse(request);
    } catch (error) {
      if (attempt === maxRetries) throw error;

      // Exponential backoff for rate limits and server errors
      if (error.type === 'rate_limit_exceeded' || error.status >= 500) {
        const delay = Math.pow(2, attempt) * 1000; // 2s, 4s, 8s...
        await new Promise(resolve => setTimeout(resolve, delay));
        continue;
      }

      // Don't retry for client errors
      if (error.status >= 400 && error.status < 500) {
        throw error;
      }
    }
  }
}
```

Rate Limiting

Understanding OpenAI Rate Limits

OpenAI implements rate limiting based on:
- **Requests per minute (RPM)**
- **Tokens per minute (TPM)**
- **Requests per day (RPD)**

Rate Limit Headers

```
const response = await openaiIntegration.generateResponse(request);

// Check rate limit information
const rateLimitInfo = {
  requestsRemaining: response.headers['x-ratelimit-remaining-requests'],
  tokensRemaining: response.headers['x-ratelimit-remaining-tokens'],
  resetTime: response.headers['x-ratelimit-reset-requests'],
  retryAfter: response.headers['retry-after']
};

console.log('Rate limit status:', rateLimitInfo);
```

Rate Limit Management

```
class RateLimitManager {
  private requests: number[] = [];
  private tokens: number[] = [];

  async checkRateLimit(estimatedTokens: number): Promise<boolean> {
    const now = Date.now();
    const oneMinuteAgo = now - 60000;

    // Clean old requests
    this.requests = this.requests.filter(time => time > oneMinuteAgo);
    this.tokens = this.tokens.filter(time => time > oneMinuteAgo);

    // Check limits (example limits)
    const requestsPerMinute = 500;
    const tokensPerMinute = 10000;

    if (this.requests.length >= requestsPerMinute) {
      return false; // Would exceed request limit
    }

    if (this.tokens.length + estimatedTokens > tokensPerMinute) {
      return false; // Would exceed token limit
    }

    return true;
  }

  recordRequest(tokenCount: number): void {
    const now = Date.now();
    this.requests.push(now);
    this.tokens.push(now);
  }
}
```

Cost Management

Cost Calculation

```typescript
interface CostBreakdown {
  inputTokens: number;
  outputTokens: number;
  inputCost: number;
  outputCost: number;
  totalCost: number;
  model: string;
}

function calculateCost(usage: TokenUsage, model: string): CostBreakdown {
  const pricing = {
    'gpt-4o': { input: 0.005, output: 0.015 },
    'gpt-4o-mini': { input: 0.00015, output: 0.0006 },
    'gpt-4-turbo': { input: 0.01, output: 0.03 },
    'gpt-3.5-turbo': { input: 0.0005, output: 0.0015 }
  };

  const modelPricing = pricing[model] || pricing['gpt-4o-mini'];
  const inputCost = (usage.promptTokens / 1000) * modelPricing.input;
  const outputCost = (usage.completionTokens / 1000) * modelPricing.output;

  return {
    inputTokens: usage.promptTokens,
    outputTokens: usage.completionTokens,
    inputCost,
    outputCost,
    totalCost: inputCost + outputCost,
    model
  };
}
```

Budget Controls

```
class BudgetManager {
  private dailyBudget = 100; // $100 per day
  private currentSpend = 0;
  private budgetResetTime = this.getNextMidnight();

  async checkBudget(estimatedCost: number): Promise<boolean> {
    this.resetBudgetIfNeeded();

    if (this.currentSpend + estimatedCost > this.dailyBudget) {
      console.warn(`Budget exceeded: $${this.currentSpend + estimatedCost} > $${this.da
ilyBudget}`);
      return false;
    }

    return true;
  }

  recordSpend(cost: number): void {
    this.currentSpend += cost;
  }

  private resetBudgetIfNeeded(): void {
    if (Date.now() > this.budgetResetTime) {
      this.currentSpend = 0;
      this.budgetResetTime = this.getNextMidnight();
    }
  }

  private getNextMidnight(): number {
    const tomorrow = new Date();
    tomorrow.setDate(tomorrow.getDate() + 1);
    tomorrow.setHours(0, 0, 0, 0);
    return tomorrow.getTime();
  }
}
```

Best Practices

Prompt Engineering

1. **Be Specific and Clear**:

```
// Poor
const badPrompt = "Write about AI";

// Better
const goodPrompt = `Write a 200-word explanation of machine learning for a non-
technical audience.
Include:
- Definition of machine learning
- One concrete example
- Key benefits
- Common applications`;
```

1. **Use System Messages**:

```
const messages = [
  {
    role: 'system',
    content:
'You are an expert data scientist. Provide accurate, technical information in a clear,
accessible way.'
  },
  {
    role: 'user',
    content: 'Explain neural networks'
  }
];
```

1. **Provide Examples**:

```
const fewShotPrompt = `
Extract key information from product reviews in JSON format.

Example:
Review: "This phone has great battery life but the camera is disappointing."
Output: {"sentiment": "mixed", "pros": ["battery life"], "cons": ["camera quality"]}

Review: "Amazing laptop! Fast processor and beautiful display. Highly recommended."
Output: {"sentiment": "positive", "pros": ["processor speed", "display quality"],
"cons": []}

Now extract from: "The software is buggy but customer service was helpful."
`;
```

Model Selection Strategy

```
function selectOptimalModel(request: AIRequest): string {
  // Simple tasks -> cost-effective models
  if (request.maxTokens < 100 && !request.requiresReasoning) {
    return 'gpt-4o-mini';
  }

  // Complex analysis -> powerful models
  if (request.requiresReasoning || request.complexity === 'high') {
    return 'gpt-4o';
  }

  // Creative content -> balanced model
  if (request.type === 'creative') {
    return 'gpt-4-turbo';
  }

  // Default choice
  return 'gpt-4o-mini';
}
```

Performance Optimization

1. **Use Appropriate Max Tokens**:

```
// Don't request more tokens than needed
const request = {
  model: 'gpt-4o-mini',
  messages: [{ role: 'user', content: 'Summarize this in 2 sentences' }],
  maxTokens: 100 // Sufficient for 2 sentences
};
```

1. **Implement Caching**:

```
const responseCache = new Map();

async function getCachedResponse(request: AIRequest): Promise<AIResponse> {
  const cacheKey = JSON.stringify(request);

  if (responseCache.has(cacheKey)) {
    return responseCache.get(cacheKey);
  }

  const response = await openaiIntegration.generateResponse(request);
  responseCache.set(cacheKey, response);

  return response;
}
```

1. **Batch Similar Requests**:

```
// Process multiple similar requests efficiently
const batchRequests = [
  { content: 'Translate "Hello" to French' },
  { content: 'Translate "Goodbye" to French' },
  { content: 'Translate "Thank you" to French' }
];

// Better: Single request with all translations
const batchRequest = {
  model: 'gpt-4o-mini',
  messages: [{
    role: 'user',
    content: `Translate these phrases to French:
    1. Hello
    2. Goodbye
    3. Thank you

    Format as JSON: {"1": "translation", "2": "translation", "3": "translation"}`
  }]
};
```

Troubleshooting

Common Issues

1. Authentication Errors

```
Error: 401 Unauthorized
```

Solution: Check API key configuration

```
// Verify API key is set
if (!process.env.OPENAI_API_KEY) {
  throw new Error('OPENAI_API_KEY environment variable is required');
}

// Test authentication
const response = await fetch('https://api.openai.com/v1/models', {
  headers: {
    'Authorization': `Bearer ${process.env.OPENAI_API_KEY}`
  }
});
```

2. Rate Limit Errors

```
Error: 429 Rate limit exceeded
```

Solution: Implement exponential backoff

```
async function handleRateLimit(error: any): Promise<void> {
  if (error.status === 429) {
    const retryAfter = parseInt(error.headers['retry-after']) || 60;
    console.log(`Rate limited. Waiting ${retryAfter} seconds...`);
    await new Promise(resolve => setTimeout(resolve, retryAfter * 1000));
  }
}
```

3. Context Length Errors

```
Error: This model's maximum context length is X tokens
```

Solution: Implement token counting and truncation

```
function truncateToTokenLimit(text: string, maxTokens: number): string {
  // Rough estimation: 1 token ≈ 4 characters
  const estimatedTokens = text.length / 4;

  if (estimatedTokens <= maxTokens) {
    return text;
  }

  const maxChars = maxTokens * 4 * 0.9; // 90% safety margin
  return text.substring(0, maxChars);
}
```

4. Model Not Found Errors

```
Error: The model 'gpt-5' does not exist
```

Solution: Validate model names

```
const validModels = [
  'gpt-4o', 'gpt-4o-mini', 'gpt-4-turbo', 'gpt-3.5-turbo'
];

function validateModel(model: string): void {
  if (!validModels.includes(model)) {
    throw new Error(`Invalid model: ${model}. Valid models: ${validModels.join(', ')}`)
;
  }
}
```

Debug Mode

Enable debug logging to troubleshoot issues:

```
const openaiIntegration = new OpenAIAdvancedIntegration({
  apiKey: process.env.OPENAI_API_KEY,
  debug: true,
  logLevel: 'verbose'
});

// This will log:
// - Request details
// - Response metadata
// - Error information
// - Performance metrics
```

Health Check

```
async function performHealthCheck(): Promise<boolean> {
  try {
    const response = await openaiIntegration.generateResponse({
      model: 'gpt-4o-mini',
      messages: [{ role: 'user', content: 'Hello' }],
      maxTokens: 5
    });

    return response.success;
  } catch (error) {
    console.error('OpenAI health check failed:', error);
    return false;
  }
}

// Run health check every 5 minutes
setInterval(performHealthCheck, 5 * 60 * 1000);
```

For additional support and advanced use cases, refer to the main AI Routing documentation (../ai-routing/README.md) and troubleshooting guide (../troubleshooting/ai-routing.md).

Last updated: August 2025
Version: 3.15.0