AI Function Calling Guide

This guide teaches you how to use OpenAI's function calling feature to enable AI models to interact with external tools and retrieve structured data.

Table of Contents

What is Function Calling?

Function calling (also called "tool use") allows AI models to request that you execute specific functions to retrieve information or perform actions.

How It Works

1. **You define available functions** - Tell the AI what functions exist and what they do
2. **AI decides when to use them** - Based on the conversation, AI chooses which function to call
3. **AI returns function call requests** - Instead of text, AI responds with function names and arguments
4. **You execute the functions** - Run the functions client-side with the provided arguments
5. **You send results back** (optional) - Give the function results to the AI for further processing

Key Concept: Schema vs Implementation

Important: When calling the OpenAI API, you only send the function **schema** (name, description, parameters), not the actual code implementation. The implementation lives in your client code.

```
// ✅ What you send to OpenAI (schema only)
{
  name: "scan_environment",
  description: "Scan the drone's surroundings",
  parameters: { type: "object", properties: {} }
}

// ✅ What you implement in your code (actual function)
function scan_environment() {
  return "Scanning complete";
}
```

Function Schema Structure

Function schemas follow the OpenAI function calling format, which is based on JSON Schema.

Basic Function Schema

```
{
  type: "function",
  function: {
    name: "function_name",           // Function identifier (no spaces)
    description: "What this does",   // Helps AI decide when to use it
    parameters: {                    // JSON Schema for parameters
      type: "object",
      properties: {},                // Define parameters here
      required: []                   // List required parameters
    }
  }
}
```

Function with No Parameters

```
{
  type: "function",
  function: {
    name: "scan_environment",
    description: "Scan the drone's current surroundings for clues",
    parameters: {
      type: "object",
      properties: {}  // Empty = no parameters needed
    }
  }
}
```

Function with Parameters

```
{
  type: "function",
  function: {
    name: "analyze_evidence",
    description: "Analyze a specific piece of evidence at the crime scene",
    parameters: {
```

```
      type: "object",
      properties: {
        evidence_id: {
          type: "string",
          description: "The ID of the evidence to analyze"
        }
      },
      required: ["evidence_id"]  // This parameter is mandatory
    }
  }
}
```

Multiple Parameters

```
{
  type: "function",
  function: {
    name: "decode_sensor_data",
    description: "Decode encrypted sensor data from the drone",
    parameters: {
      type: "object",
      properties: {
        data_type: {
          type: "string",
          description: "Type of sensor data (thermal, optical, acoustic)",
          enum: ["thermal", "optical", "acoustic"]  // Restrict to specific values
        },
        encoded_value: {
          type: "string",
          description: "The encoded sensor reading to decode"
        }
      },
      required: ["data_type", "encoded_value"]
    }
  }
```

```
}
```

Making API Requests

Basic Request Structure

```
const response = await fetch('/api/v1/ai/chat', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json'
  },
  body: JSON.stringify({
    model: 'gpt-4o-mini',
    messages: [
      {
        role: 'user',
        content: 'Scan the environment and tell me what you find'
      }
    ],
    tools: [
      // Your function schemas go here
      {
        type: "function",
        function: {
          name: "scan_environment",
          description: "Scan surroundings for clues",
          parameters: {
            type: "object",
            properties: {}
          }
        }
      }
    ]
  })
});
```

```
const data = await response.json();
```

Complete Request Example

```
const tools = [
  {
    type: "function",
    function: {
      name: "scan_environment",
      description: "Scan the drone's surroundings",
      parameters: {
        type: "object",
        properties: {}
      }
    }
  },
  {
    type: "function",
    function: {
      name: "analyze_evidence",
      description: "Analyze evidence at the scene",
      parameters: {
        type: "object",
        properties: {
          evidence_id: {
            type: "string",
            description: "ID of evidence to analyze"
          }
        },
        required: ["evidence_id"]
      }
    }
  },
  {
    type: "function",
```

```
    function: {
      name: "decode_sensor_data",
      description: "Decode hex-encoded sensor readings",
      parameters: {
        type: "object",
        properties: {
          encoded_data: {
            type: "string",
            description: "Hex-encoded sensor data"
          }
        },
        required: ["encoded_data"]
      }
    }
  }
];

const response = await fetch('/api/v1/ai/chat', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json'
  },
  body: JSON.stringify({
    model: 'gpt-4o-mini',
    messages: [
      {
        role: 'user',
        content: 'I need you to scan the environment and analyze any evidence you find.'
      }
    ],
    tools: tools
  })
});

const data = await response.json();
```

```
console.log(data);
```

Handling Function Call Responses

When the AI wants to call a function, it responds with `tool_calls` instead of regular text content.

Response Structure

```
{
  "choices": [
    {
      "message": {
        "role": "assistant",
        "content": null,  // No text content when calling functions
        "tool_calls": [
          {
            "id": "call_abc123",
            "type": "function",
            "function": {
              "name": "scan_environment",
              "arguments": "{}"  // JSON string of arguments
            }
          }
        ]
      }
    }
  ]
}
```

Extracting Function Calls

```
const response = await fetch('/api/v1/ai/chat', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({
    model: 'gpt-4o-mini',
    messages: [{ role: 'user', content: 'Scan the area' }],
```

```
    tools: tools
  })
});

const data = await response.json();
const message = data.choices[0].message;

// Check if AI wants to call functions
if (message.tool_calls) {
  console.log('AI is requesting function calls:');

  for (const toolCall of message.tool_calls) {
    const functionName = toolCall.function.name;
    const functionArgs = JSON.parse(toolCall.function.arguments);

    console.log(`Function: ${functionName}`);
    console.log(`Arguments:`, functionArgs);

    // Now execute the function with these arguments
    const result = executeFunction(functionName, functionArgs);
    console.log(`Result:`, result);
  }
} else {
  // Regular text response
  console.log('AI response:', message.content);
}
```

Executing Functions Based on Name

```
// Define your function implementations
const functions = {
  scan_environment: () => {
    return { status: "scan complete", data: "warehouse detected" };
  },
```

```
  analyze_evidence: (args) => {
    const evidenceId = args.evidence_id;
    return { evidence_id: evidenceId, type: "weapon residue" };
  },

  decode_sensor_data: (args) => {
    const encoded = args.encoded_data;
    // Decode hex to string
    const decoded = hexToString(encoded);
    return { decoded_value: decoded };
  }
};

// Execute function dynamically
function executeFunction(functionName, args) {
  if (functions[functionName]) {
    return functions[functionName](args);
  } else {
    throw new Error(`Unknown function: ${functionName}`);
  }
}

// Usage
for (const toolCall of message.tool_calls) {
  const name = toolCall.function.name;
  const args = JSON.parse(toolCall.function.arguments);
  const result = executeFunction(name, args);
  console.log(`${name} returned:`, result);
}
```

Decoding Encoded Data

Function arguments often contain encoded data that needs to be decoded to extract the real information.

Hex Decoding

```
// Decode hex string to regular string
```

```
function hexToString(hex) {
  let str = '';
  for (let i = 0; i < hex.length; i += 2) {
    const hexByte = hex.substr(i, 2);
    str += String.fromCharCode(parseInt(hexByte, 16));
  }
  return str;
}

// Usage
const hexData = "57415245484F555345";  // "WAREHOUSE" in hex
const decoded = hexToString(hexData);
console.log(decoded);  // "WAREHOUSE"
```

Base64 Decoding

```
// Decode Base64 to string
const base64Data = "MjAyNS0wMS0xNVQxNDozMDowMFo=";
const decoded = atob(base64Data);
console.log(decoded);  // "2025-01-15T14:30:00Z"
```

Processing Function Arguments with Encoded Data

```
const functions = {
  decode_sensor_data: (args) => {
    const hexData = args.encoded_data;

    // Decode hex to string
    const decoded = hexToString(hexData);

    console.log('Decoded sensor data:', decoded);
    return decoded;
  }
};

// When AI calls this function with hex data
```

```
const toolCall = {
  function: {
    name: "decode_sensor_data",
    arguments: '{"encoded_data":"57415245484F555345"}'
  }
};

const args = JSON.parse(toolCall.function.arguments);
const result = functions.decode_sensor_data(args);
// Result: "WAREHOUSE"
```

Complete Examples

Example 1: Single Function Call

```
// Define function schema
const tools = [
  {
    type: "function",
    function: {
      name: "scan_environment",
      description: "Scan the drone's surroundings",
      parameters: {
        type: "object",
        properties: {}
      }
    }
  }
];

// Make API request
const response = await fetch('/api/v1/ai/chat', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({
    model: 'gpt-4o-mini',
```

```
    messages: [
      { role: 'user', content: 'What do you see around you?' }
    ],
    tools: tools
  })
});


const data = await response.json();
const message = data.choices[0].message;


// Handle function call
if (message.tool_calls) {
  const toolCall = message.tool_calls[0];
  const functionName = toolCall.function.name;


  console.log(`AI wants to call: ${functionName}`);


  // Execute function
  if (functionName === 'scan_environment') {
    const result = scan_environment();
    console.log('Scan result:', result);
  }
}
```

Example 2: Multiple Function Calls with Hex Decoding

```
// Helper function to decode hex
function hexToString(hex) {
  let str = '';
  for (let i = 0; i < hex.length; i += 2) {
    str += String.fromCharCode(parseInt(hex.substr(i, 2), 16));
  }
  return str;
}
```

```
// Define tools
const tools = [
  {
    type: "function",
    function: {
      name: "scan_environment",
      description: "Scan surroundings",
      parameters: { type: "object", properties: {} }
    }
  },
  {
    type: "function",
    function: {
      name: "decode_sensor_data",
      description: "Decode hex sensor data",
      parameters: {
        type: "object",
        properties: {
          encoded_data: {
            type: "string",
            description: "Hex-encoded data"
          }
        },
        required: ["encoded_data"]
      }
    }
  }
];

// Define function implementations
const functions = {
  scan_environment: () => {
    return "Scan complete - building type detected";
  },
```

```
  decode_sensor_data: (args) => {
    return hexToString(args.encoded_data);
  }
};

// Make request
const response = await fetch('/api/v1/ai/chat', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({
    model: 'gpt-4o-mini',
    messages: [
      { role: 'user', content: 'Scan the area and decode any sensor data you find' }
    ],
    tools: tools
  })
});

const data = await response.json();
const message = data.choices[0].message;

// Process all function calls
if (message.tool_calls) {
  console.log(`AI is calling ${message.tool_calls.length} functions:\n`);

  const results = [];

  for (const toolCall of message.tool_calls) {
    const name = toolCall.function.name;
    const args = JSON.parse(toolCall.function.arguments);

    console.log(`Executing: ${name}`);
    console.log(`Arguments:`, args);

    // Execute function
```

```
    const result = functions[name](args);
    console.log(`Result: ${result}\n`);

    results.push({ function: name, result: result });
  }

  console.log('All results:', results);
}
```

Example 3: Complete Workflow with Evidence Analysis

```
function hexToString(hex) {
  let str = '';
  for (let i = 0; i < hex.length; i += 2) {
    str += String.fromCharCode(parseInt(hex.substr(i, 2), 16));
  }
  return str;
}

// Define all tools for the drone
const tools = [
  {
    type: "function",
    function: {
      name: "scan_environment",
      description: "Scan the drone's surroundings to identify the location type",
      parameters: {
        type: "object",
        properties: {}
      }
    }
  },
  {
    type: "function",
    function: {
```

```
      name: "analyze_evidence",
      description: "Analyze evidence found at the crime scene",
      parameters: {
        type: "object",
        properties: {
          evidence_id: {
            type: "string",
            description: "The identifier for the evidence sample"
          }
        },
        required: ["evidence_id"]
      }
    }
  },
  {
    type: "function",
    function: {
      name: "decode_drone_log",
      description: "Decode encrypted log data from the drone's memory",
      parameters: {
        type: "object",
        properties: {
          encoded_data: {
            type: "string",
            description: "Hex-encoded log entry"
          }
        },
        required: ["encoded_data"]
      }
    }
  }
];

// Implement the functions
```

```
const functions = {
  scan_environment: () => {
    return { status: "complete", location_detected: true };
  },

  analyze_evidence: (args) => {
    return {
      evidence_id: args.evidence_id,
      analysis: "residue detected",
      weapon_type: "identified"
    };
  },

  decode_drone_log: (args) => {
    const decoded = hexToString(args.encoded_data);
    return {
      original: args.encoded_data,
      decoded: decoded
    };
  }
};

// Send request to AI
const response = await fetch('/api/v1/ai/chat', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({
    model: 'gpt-4o-mini',
    messages: [
      {
        role: 'user',
        content: 'Activate all drone systems. Scan the environment, analyze any evidence you detect, and decode your log
data.'
      }
    ],
```

```
    tools: tools
  })
});

const data = await response.json();
const message = data.choices[0].message;

// Process function calls
const findings = {};

if (message.tool_calls) {
  for (const toolCall of message.tool_calls) {
    const name = toolCall.function.name;
    const args = JSON.parse(toolCall.function.arguments);

    console.log(`\n🔧 Executing: ${name}`);

    const result = functions[name](args);
    findings[name] = result;

    console.log('Result:', result);
  }

  console.log('\n🔍 All Findings:');
  console.log(findings);
}
```

Tips & Best Practices

1. **Write clear descriptions** - The AI uses descriptions to decide when to call functions
2. **Use descriptive parameter names** - `evidence_id` is better than `id`
3. **Define required parameters** - Use the `required` array to specify mandatory params
4. **Test with simple prompts** - Start with "Call function X" to verify your setup
5. **Handle missing function calls** - Check if `message.tool_calls` exists before accessing it
6. **Log everything** - Use console.log to see what the AI is requesting
7. **Implement all defined functions** - Don't define functions you can't execute

Common Pitfalls

❌ **Sending function implementation instead of schema**

```
// Wrong - don't send actual code
tools: [
  function scan_environment() { return "scan"; }
]
```

✅ **Correct - send schema only**

```
tools: [
  {
    type: "function",
    function: {
      name: "scan_environment",
      description: "Scan surroundings",
      parameters: { type: "object", properties: {} }
    }
  }
]
```

❌ **Not parsing arguments JSON string**

```
const args = toolCall.function.arguments; // Still a string!
const evidenceId = args.evidence_id; // undefined
```

✅ **Correct - parse JSON first**

```
const args = JSON.parse(toolCall.function.arguments);
const evidenceId = args.evidence_id; // Works!
```

❌ **Forgetting to decode hex/base64 data**

```
console.log(args.encoded_data); // Prints hex gibberish
```

✅ **Correct - decode first**

```
const decoded = hexToString(args.encoded_data);
console.log(decoded); // Prints actual string
```

Quick Reference

```
// Function schema template
{
  type: "function",
  function: {
    name: "function_name",
    description: "What it does",
    parameters: {
      type: "object",
      properties: {
        param_name: {
          type: "string",
          description: "What this parameter is"
        }
      },
      required: ["param_name"]
    }
  }
}

// API request
const response = await fetch('/api/v1/ai/chat', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({
    model: 'gpt-4o-mini',
    messages: [{ role: 'user', content: 'Your prompt' }],
    tools: tools  // Array of function schemas
  })
});

// Handle response
const data = await response.json();
if (data.choices[0].message.tool_calls) {
```

```
  for (const call of data.choices[0].message.tool_calls) {
    const name = call.function.name;
    const args = JSON.parse(call.function.arguments);
    const result = functions[name](args);
  }
}


// Decode hex
function hexToString(hex) {
  let str = '';
  for (let i = 0; i < hex.length; i += 2) {
    str += String.fromCharCode(parseInt(hex.substr(i, 2), 16));
  }
  return str;
}
```

Resources

- OpenAI Function Calling Guide
- JSON Schema Documentation
- MDN: JSON.parse()
- MDN: String.fromCharCode()