AI 联网搜索功能实现方案

根据你的需求，我将提供一个完整的实现方案，集成 LangChain 的多个功能模块，实现类似 Monica 的联网搜索功能。

📋 实现步骤总览

第一阶段：环境准备

1. 安装依赖包
2. 配置环境变量
3. 创建工具文件

第二阶段：后端实现

1. 创建 Bocha 搜索工具
2. 创建 LangChain Agent
3. 创建联网搜索 API 路由
4. 集成到现有聊天 API

第三阶段：前端实现

1. 添加联网搜索按钮
2. 修改输入区域组件
3. 修改聊天区域组件
4. 更新消息显示逻辑

🔧 详细实现步骤

步骤 1：安装依赖包

在项目根目录执行：

```
复制
npm install @langchain/community @langchain/core langchain axios
```

或者在 `package.json` 中添加：

```
复制{
  "dependencies": {
    "@langchain/community": "^0.0.20",
    "@langchain/core": "^0.1.10",
    "langchain": "^0.1.10",
    "axios": "^1.6.0"
  }
}
```

步骤 2：配置环境变量

在 `.env.local` 文件中添加：

```
复制# 现有配置
OPENAI_API_KEY=your-openrouter-api-key
NEXTAUTH_URL=http://localhost:3000
DATABASE_URL=your-database-url

# 新增：Bocha API Key
BOCHA_API_KEY=your-bocha-api-key
```

步骤 3：创建 LangChain 工具文件

📂 新建文件：`lib/langchain/tools.js`

```
复制/**
 * ============================================================================
 * LangChain 工具集 (lib/langchain/tools.js)
 * ============================================================================
 *
 * 文件作用：
 *   定义 LangChain 使用的工具（Tools），包括网页搜索、计算器等
 *
 * 主要功能：
 *   1. Bocha Web Search 工具（联网搜索）
 *   2. Calculator 工具（数学计算）
 *   3. 可扩展其他工具（Wikipedia、天气查询等）
 *
 * 技术栈：
 *   - LangChain Tools
 *   - Axios（HTTP 请求）
 *   - Bocha API
 *
 * ============================================================================
 */

import { DynamicStructuredTool } from "@langchain/core/tools";
import { z } from "zod";
import axios from "axios";
```

```
/**
 * Bocha Web Search 工具
 *
 * 功能：使用 Bocha API 进行网页搜索
 *
 * 参数：
 *   - query: 搜索关键词
 *   - count: 返回结果数量（默认 10）
 *
 * 返回：
 *   格式化的搜索结果，包括标题、URL、摘要、发布时间等
 */
export const bochaWebSearchTool = new DynamicStructuredTool({
  name: "bocha_web_search",
  description: `使用 Bocha Web Search API 进行互联网搜索。
    适用场景：
    - 查询最新新闻、事件
    - 搜索实时信息（股票、天气、赛事等）
    - 查找网页内容、文档、报告
    输入：搜索关键词字符串
    输出：搜索结果列表（标题、URL、摘要、发布时间）`,

  schema: z.object({
    query: z.string().describe("搜索关键词"),
    count: z.number().optional().default(10).describe("返回结果数量，默认 10"),
  }),

  func: async ({ query, count = 10 }) => {
    try {
      const response = await axios.post(
        'https://api.bochaai.com/v1/web-search',
        {
          query,
          freshness: "noLimit", // 时间范围：oneDay, oneWeek, oneMonth, oneYear, noLimit
          summary: true,        // 返回长文本摘要
          count,
        },
        {
          headers: {
            'Authorization': `Bearer ${process.env.BOCHA_API_KEY}`,
            'Content-Type': 'application/json',
          },
        }
      );

      // 解析响应
      const data = response.data;
      if (data.code !== 200 || !data.data) {
        return `搜索失败：${data.msg || '未知错误'}`;
      }

      const webpages = data.data.webPages?.value || [];
```

```
      if (webpages.length === 0) {
        return "未找到相关结果。";
      }

      // 格式化搜索结果
      let formattedResults = `找到 ${webpages.length} 条搜索结果：\n\n`;
      webpages.forEach((page, idx) => {
        formattedResults += `[${idx + 1}] ${page.name}\n`;
        formattedResults += `   URL: ${page.url}\n`;
        formattedResults += `   摘要: ${page.summary}\n`;
        formattedResults += `   来源: ${page.siteName} | 发布时间:
${page.dateLastCrawled}\n\n`;
      });

      return formattedResults.trim();
    } catch (error) {
      console.error('Bocha 搜索失败:', error);
      return `搜索 API 请求失败：${error.message}`;
    }
  },
});

/**
 * Calculator 工具（可选）
 *
 * 功能：执行数学计算
 */
export const calculatorTool = new DynamicStructuredTool({
  name: "calculator",
  description: "执行数学计算，输入数学表达式，返回计算结果",

  schema: z.object({
    expression: z.string().describe("数学表达式，如 '2 + 2' 或 '10 * 5'"),
  }),

  func: async ({ expression }) => {
    try {
      // 使用 eval（生产环境建议使用 math.js 等安全库）
      const result = eval(expression);
      return `计算结果：${expression} = ${result}`;
    } catch (error) {
      return `计算失败：${error.message}`;
    }
  },
});
```

步骤 4：创建 LangChain Agent

📁 新建文件：`lib/langchain/agent.js`

```
复制/**
 * ============================================================================
 * LangChain Agent 配置 (lib/langchain/agent.js)
 * ============================================================================
 *
 * 文件作用：
 *   创建和配置 LangChain Agent，集成工具和记忆功能
 *
 * 主要功能：
 *   1. 初始化 Agent（ReAct 类型）
 *   2. 集成工具（搜索、计算器等）
 *   3. 管理对话记忆（BufferMemory）
 *   4. 流式输出支持
 *
 * 技术栈：
 *   - LangChain Agents
 *   - ChatOpenAI（通过 OpenRouter）
 *   - BufferMemory（对话记忆）
 *
 * ============================================================================
 */

import { ChatOpenAI } from "@langchain/openai";
import { AgentExecutor, createStructuredChatAgent } from "langchain/agents";
import { ChatPromptTemplate, MessagesPlaceholder } from "@langchain/core/prompts";
import { BufferMemory } from "langchain/memory";
import { bochaWebSearchTool, calculatorTool } from "./tools";

/**
 * 创建 LangChain Agent
 *
 * 参数：
 *   - model: AI 模型名称（如 'gpt-4o'）
 *   - chatHistory: 历史消息数组（用于上下文）
 *
 * 返回：
 *   AgentExecutor 实例
 */
export async function createWebSearchAgent(model = "openai/gpt-4o", chatHistory = []) {
  // ============================================================================
  // 1. 初始化 LLM（通过 OpenRouter）
  // ============================================================================
  const llm = new ChatOpenAI({
    modelName: model,
    openAIApiKey: process.env.OPENAI_API_KEY,
    configuration: {
      baseURL: "https://openrouter.ai/api/v1",
    },
```

```
    temperature: 0.7,
    streaming: true, // 启用流式输出
  });

  // ============================================================================
  // 2. 定义工具列表
  // ============================================================================
  const tools = [
    bochaWebSearchTool,  // 网页搜索工具
    calculatorTool,      // 计算器工具（可选）
  ];

  // ============================================================================
  // 3. 创建 Prompt 模板
  // ============================================================================
  const prompt = ChatPromptTemplate.fromMessages([
    [
      "system",
      `你是一个智能助手，可以使用工具来回答问题。

可用工具：
{tools}

工具使用格式：
\`\`\`json
{{
  "action": "工具名称",
  "action_input": "工具参数"
}}
\`\`\`

回答要求：
1. 如果问题需要最新信息，使用 bocha_web_search 工具搜索
2. 如果问题涉及计算，使用 calculator 工具
3. 搜索结果要包含来源引用（URL）
4. 回答要详细、准确、有条理
5. 使用 Markdown 格式美化输出

工具名称：{tool_names}`,
    ],
    new MessagesPlaceholder("chat_history"),
    ["human", "{input}"],
    new MessagesPlaceholder("agent_scratchpad"),
  ]);

  // ============================================================================
  // 4. 创建 Agent
  // ============================================================================
  const agent = await createStructuredChatAgent({
    llm,
    tools,
    prompt,
  });
```

```
  // ========================================================================
  // 5. 创建 Memory（对话记忆）
  // ========================================================================
  const memory = new BufferMemory({
    memoryKey: "chat_history",
    returnMessages: true,
    inputKey: "input",
    outputKey: "output",
  });

  // 加载历史消息到 Memory
  for (const msg of chatHistory) {
    if (msg.role === "user") {
      await memory.chatHistory.addUserMessage(msg.content);
    } else if (msg.role === "assistant") {
      await memory.chatHistory.addAIChatMessage(msg.content);
    }
  }

  // ========================================================================
  // 6. 创建 AgentExecutor
  // ========================================================================
  const agentExecutor = new AgentExecutor({
    agent,
    tools,
    memory,
    verbose: true, // 开发时打印调试信息
    maxIterations: 5, // 最大工具调用次数
    returnIntermediateSteps: true, // 返回中间步骤（用于显示搜索过程）
  });

  return agentExecutor;
}
```

步骤 5：创建联网搜索 API 路由

📁 新建文件：`app/api/chat-web-search/route.js`

```
复制/**
 * ========================================================================
 * 联网搜索聊天 API (app/api/chat-web-search/route.js)
 * ========================================================================
 *
 * 文件作用：
 *   处理带联网搜索功能的聊天请求
 *
 * 主要功能：
 *   1. 接收用户消息和历史记录
 *   2. 调用 LangChain Agent 进行搜索和回答
 *   3. 返回流式响应（SSE）
```

```
 *   4. 显示中间步骤（搜索过程）
 *
 * 路由：POST /api/chat-web-search
 *
 * 请求体：
 *   {
 *     messages: Array<{role, content}>,
 *     model: string
 *   }
 *
 * 响应：
 *   - Content-Type: text/event-stream
 *   - 格式：data: {"content": "...", "type": "text|tool"}
 *
 * ============================================================================
 */

import { createWebSearchAgent } from "@/lib/langchain/agent";
import { NextResponse } from "next/server";

export async function POST(req) {
  try {
    // ========================================================================
    // 1. 解析请求体
    // ========================================================================
    const { messages, model } = await req.json();

    if (!messages || messages.length === 0) {
      return NextResponse.json(
        { error: "消息列表不能为空" },
        { status: 400 }
      );
    }

    // 提取最后一条用户消息
    const lastMessage = messages[messages.length - 1];
    const userInput = lastMessage.content;

    // 提取历史消息（用于上下文）
    const chatHistory = messages.slice(0, -1);

    // ========================================================================
    // 2. 创建 Agent
    // ========================================================================
    const agent = await createWebSearchAgent(model, chatHistory);

    // ========================================================================
    // 3. 创建流式响应
    // ========================================================================
    const encoder = new TextEncoder();
    const stream = new ReadableStream({
      async start(controller) {
        try {
```

```
      // ============================================================
      // 3.1 执行 Agent（流式输出）
      // ============================================================
      const result = await agent.invoke(
        { input: userInput },
        {
          callbacks: [
            {
              // 处理 LLM 输出（文本内容）
              handleLLMNewToken(token) {
                const data = JSON.stringify({
                  content: token,
                  type: "text",
                });
                controller.enqueue(encoder.encode(`data: ${data}\n\n`));
              },

              // 处理工具调用（搜索过程）
              handleToolStart(tool, input) {
                const data = JSON.stringify({
                  content: `\n🔍 正在使用工具：${tool.name}\n`,
                  type: "tool",
                });
                controller.enqueue(encoder.encode(`data: ${data}\n\n`));
              },

              // 处理工具输出（搜索结果）
              handleToolEnd(output) {
                const data = JSON.stringify({
                  content: `\n✅ 工具执行完成\n`,
                  type: "tool",
                });
                controller.enqueue(encoder.encode(`data: ${data}\n\n`));
              },
            },
          ],
        }
      );

      // ============================================================
      // 3.2 发送最终结果
      // ============================================================
      if (result.output) {
        const data = JSON.stringify({
          content: result.output,
          type: "final",
        });
        controller.enqueue(encoder.encode(`data: ${data}\n\n`));
      }

      // 关闭流
      controller.close();
    } catch (error) {
```

```
          console.error("Agent 执行失败:", error);
          const data = JSON.stringify({
            content: `\n❌ 错误: ${error.message}\n`,
            type: "error",
          });
          controller.enqueue(encoder.encode(`data: ${data}\n\n`));
          controller.close();
        }
      },
    });

    // ============================================================================
    // 4. 返回流式响应
    // ============================================================================
    return new Response(stream, {
      headers: {
        "Content-Type": "text/event-stream",
        "Cache-Control": "no-cache",
        "Connection": "keep-alive",
      },
    });
  } catch (error) {
    console.error("联网搜索 API 错误:", error);
    return NextResponse.json(
      { error: error.message || "服务器错误" },
      { status: 500 }
    );
  }
}
```

步骤 6：修改前端 - 输入区域组件

📂 修改文件：`app/components/chat/InputArea.js`

在现有代码中添加联网搜索按钮：

```
复制// ============================================================================
// 在 InputArea 组件中添加联网搜索状态
// ============================================================================

export default function InputArea({
  onSendMessage,
  currentModel,
  isGenerating,
  onStopGeneration
}) {
  const [input, setInput] = useState('')
  const [images, setImages] = useState([])
  const [isDragging, setIsDragging] = useState(false)

  // ✅ 新增：联网搜索状态
```

```
const [isWebSearchEnabled, setIsWebSearchEnabled] = useState(false)

// ... 其他代码保持不变 ...

// ✅ 修改发送消息函数，传递联网搜索标志
const handleSend = () => {
  if (isGenerating) {
    onStopGeneration()
    setTimeout(() => {
      sendMessage()
    }, 200)
    return
  }

  if (!input.trim() && images.length === 0) return

  const uploadingImages = images.filter((img) => img.uploading)
  if (uploadingImages.length > 0) {
    alert("请等待图片上传完成")
    return
  }

  const imageUrls = images.map((img) => img.serverUrl)

  // ✅ 传递 isWebSearchEnabled 标志
  onSendMessage(input.trim(), imageUrls, isWebSearchEnabled)

  setInput("")
  setImages([])
  if (textareaRef.current) {
    textareaRef.current.style.height = "auto"
  }
}

return (
  <div className="border-t border-gray-200 bg-white">
    <div className="max-w-4xl mx-auto px-4 py-4">

      {/* 图片预览区域（保持不变）*/}
      {/* ... */}

      {/* ✅ 输入框容器 - 添加联网搜索按钮 */}
      <div
        className={cn(
          "relative flex items-end gap-2 p-3 rounded-2xl border-2 transition-colors",
          isDragging
            ? "border-blue-500 bg-blue-50"
            : "border-gray-200 bg-white"
        )}
        onDragOver={(e) => {
          e.preventDefault()
          setIsDragging(true)
        }}
```

```
            onDragLeave={() => setIsDragging(false)}
            onDrop={handleDrop}
          >

            {/* 上传按钮（保持不变）*/}
            <Button
              variant="ghost"
              size="icon"
              className="h-9 w-9 shrink-0"
              onClick={() => fileInputRef.current?.click()}
              disabled={!currentModel.supportsVision}
            >
              <Paperclip className="h-5 w-5 text-gray-500" />
            </Button>

            {/* ✅ 新增：联网搜索按钮 */}
            <Button
              variant={isWebSearchEnabled ? "default" : "ghost"}
              size="icon"
              className={cn(
                "h-9 w-9 shrink-0 transition-colors",
                isWebSearchEnabled && "bg-blue-600 hover:bg-blue-700 text-white"
              )}
              onClick={() => setIsWebSearchEnabled(!isWebSearchEnabled)}
              title={isWebSearchEnabled ? "已启用联网搜索" : "点击启用联网搜索"}
            >
              <Globe className="h-5 w-5" />
            </Button>

            {/* 隐藏的文件输入框（保持不变）*/}
            <input
              ref={fileInputRef}
              type="file"
              accept="image/*"
              multiple
              className="hidden"
              onChange={handleFileChange}
            />

            {/* 文本输入框（保持不变）*/}
            <Textarea
              ref={textareaRef}
              value={input}
              onChange={handleInputChange}
              onKeyDown={handleKeyDown}
              placeholder={
                isWebSearchEnabled
                  ? "问我任何问题（已启用联网搜索）..."
                  : "问我任何问题..."
              }
              className="flex-1 min-h-[40px] max-h-[200px] resize-none border-0 focus-
visible:ring-0 focus-visible:ring-offset-0 p-0"
              rows={1}
```

```
          />

          {/* 发送/停止按钮（保持不变）*/}
          {isGenerating ? (
            <Button
              size="icon"
              className="h-9 w-9 shrink-0 rounded-full bg-red-600 hover:bg-red-700"
              onClick={onStopGeneration}
            >
              <Square className="h-4 w-4" fill="currentColor" />
            </Button>
          ) : (
            <Button
              size="icon"
              className="h-9 w-9 shrink-0 rounded-full bg-blue-600 hover:bg-blue-700"
              onClick={handleSend}
              disabled={!input.trim() && images.length === 0}
            >
              <Send className="h-4 w-4" />
            </Button>
          )}
        </div>

        {/* ✅ 提示文本 - 显示联网搜索状态 */}
        <div className="mt-2 text-xs text-gray-500 text-center">
          {isWebSearchEnabled && (
            <span className="text-blue-600 font-medium mr-2">
              🌐 联网搜索已启用
            </span>
          )}
          {currentModel.supportsVision ? (
            <>
              <ImageIcon className="inline h-3 w-3 mr-1" />
              支持图片上传 • 按 Enter 发送 • Shift+Enter 换行
            </>
          ) : (
            <>按 Enter 发送 • Shift+Enter 换行</>
          )}
        </div>
      </div>
    </div>
  )
}
```

✅ 记得在文件顶部导入 Globe 图标：

复制

```
import { Send, Paperclip, X, Image as ImageIcon, Square, Globe } from 'lucide-react'
```

步骤 7：修改 ChatLayout 组件

📂 修改文件：`app/components/chat/ChatLayout.js`

修改 `handleSendMessage` 函数以支持联网搜索：

```
复制/**
 * 发送消息（修改版 - 支持联网搜索）
 */
const handleSendMessage = async (content, images = [], useWebSearch = false) => {
  try {
    // ✅ 如果正在生成，先中断
    if (isGenerating && abortControllerRef.current) {
      abortControllerRef.current.abort()
      setIsGenerating(false)
      await new Promise(resolve => setTimeout(resolve, 100))
    }

    // 确保有当前会话
    let convId = currentConversation?.id
    if (!convId) {
      const newConv = await createConversation('新对话', selectedModel.id)
      convId = newConv.id
    }

    // 发送消息到数据库
    const aiMessageId = await sendMessage(content, images, selectedModel.id)

    // ✅ 创建新的 AbortController
    abortControllerRef.current = new AbortController()
    setIsGenerating(true)

    // ✅ 根据 useWebSearch 选择不同的 API
    const apiEndpoint = useWebSearch ? '/api/chat-web-search' : '/api/chat'

    // 调用流式 API
    const response = await fetch(apiEndpoint, {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify({
        messages: [
          ...currentMessages.map(m => ({
            role: m.role,
            content: m.content,
            images: m.images || []
          })),
          {
            role: 'user',
            content: content,
            images: images
          }
```

```
      ],
      model: selectedModel.id,
      images: images
    }),
    signal: abortControllerRef.current.signal
  })

  // 流式读取
  const reader = response.body.getReader()
  const decoder = new TextDecoder()
  let accumulatedContent = ''

  try {
    while (true) {
      const { done, value } = await reader.read()

      if (done) break
      const chunk = decoder.decode(value)
      const lines = chunk.split('\n')

      for (const line of lines) {
        if (line.startsWith('data: ')) {
          const data = JSON.parse(line.slice(6))

          // ✅ 处理不同类型的消息
          if (data.type === 'tool') {
            // 工具调用信息（搜索过程）
            accumulatedContent += data.content
            updateMessageContent(aiMessageId, accumulatedContent)
          } else if (data.type === 'text' || data.type === 'final') {
            // 正常文本内容
            accumulatedContent += data.content
            updateMessageContent(aiMessageId, accumulatedContent)
          }
        }
      }
    }

    // ✅ 流式完成，保存到数据库
    await fetch(`/api/messages/${aiMessageId}`, {
      method: 'PATCH',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ content: accumulatedContent })
    })

    // 自动生成标题
    if (currentMessages.length === 0) {
      await generateTitle(convId)
    }
  } catch (streamError) {
    // ✅ 处理中断错误
    if (streamError.name === 'AbortError') {
      console.log('用户中断了生成')
```

```
        if (accumulatedContent) {
          await fetch(`/api/messages/${aiMessageId}`, {
            method: 'PATCH',
            headers: { 'Content-Type': 'application/json' },
            body: JSON.stringify({ content: accumulatedContent + '\n\n[已停止生成]' })
          })
        }
      } else {
        throw streamError
      }
    } finally {
      // ✅ 重置生成状态
      setIsGenerating(false)
      abortControllerRef.current = null
    }
  } catch (error) {
    console.error('发送消息失败:', error)
    if (error.name !== 'AbortError') {
      alert('发送消息失败: ' + error.message)
    }
    setIsGenerating(false)
  }
}
```

步骤 8：更新模型配置

📂 修改文件：`lib/mock-data.js`

确保模型配置中包含联网搜索支持标识：

```
复制export const models = [
  {
    id: 'openai/gpt-4o',
    name: 'GPT-4o',
    provider: 'OpenAI',
    icon: '😈',
    supportsVision: true,
    supportsWebSearch: true, // ✅ 新增
    description: '最强大的模型，支持图片和联网搜索'
  },
  {
    id: 'anthropic/claude-3.5-sonnet',
    name: 'Claude 3.5 Sonnet',
    provider: 'Anthropic',
    icon: '🧠',
    supportsVision: true,
    supportsWebSearch: true, // ✅ 新增
    description: '擅长长文本和复杂推理'
  },
  // ... 其他模型
]
```

📊 涉及的 LangChain 功能清单

根据你的需求，以下是本方案使用的 LangChain 功能：

功能	使用情况	说明	文件位置
Chains（链式调用）	✅ 使用	AgentExecutor 内部使用	`lib/langchain/agent.js`
Agents（智能代理）	✅ 使用	createStructuredChatAgent	`lib/langchain/agent.js`
Memory（记忆管理）	✅ 使用	BufferMemory	`lib/langchain/agent.js`
Tools（工具调用）	✅ 使用	bochaWebSearchTool, calculatorTool	`lib/langchain/tools.js`
Output Parsers	✅ 使用	Agent 内置的结构化输出解析	`lib/langchain/agent.js`
Embeddings	❌ 未使用	本方案不涉及向量化	-
Vector Stores	❌ 未使用	本方案不涉及向量数据库	-
Document Loaders	❌ 未使用	可扩展（如 PDF 解析）	-
Text Splitters	❌ 未使用	可扩展（如长文本处理）	-
Retrievers	❌ 未使用	可扩展（如 RAG 检索）	-

📁 完整文件清单

新增文件：

1. ✅ `lib/langchain/tools.js` - 工具定义
2. ✅ `lib/langchain/agent.js` - Agent 配置
3. ✅ `app/api/chat-web-search/route.js` - 联网搜索 API

修改文件：

1. ✅ `app/components/chat/InputArea.js` - 添加联网搜索按钮
2. ✅ `app/components/chat/ChatLayout.js` - 修改发送消息逻辑
3. ✅ `lib/mock-data.js` - 更新模型配置
4. ✅ `.env.local` - 添加 BOCHA_API_KEY
5. ✅ `package.json` - 添加依赖包

🎨 UI 效果

联网搜索按钮状态：

- **未启用**：灰色地球图标（Ghost 样式）
- **已启用**：蓝色地球图标（Primary 样式）

搜索过程显示：

```
复制🔍 正在使用工具：bocha_web_search

找到 5 条搜索结果：

[1] 阿里巴巴2024年ESG报告发布
   URL: https://example.com/...
   摘要: ...
   来源: 新浪财经 | 发布时间: 2024-11-15

✅ 工具执行完成

根据搜索结果，阿里巴巴2024年ESG报告的亮点包括...
```

🧪 测试步骤

1. **启动开发服务器**：

   ```
   复制
   npm run dev
   ```

2. **测试联网搜索**：
   - 点击输入框左侧的地球图标（启用联网搜索）
   - 输入问题："阿里巴巴2024年ESG报告的亮点是什么？"
   - 观察搜索过程和结果
3. **测试普通对话**：
   - 关闭联网搜索（地球图标变灰）
   - 输入问题："你好，介绍一下自己"
   - 验证不会调用搜索工具

🚀 扩展功能建议

1. 添加更多工具

在 `lib/langchain/tools.js` 中添加：

- Wikipedia 工具
- 天气查询工具
- 股票行情工具

2. 集成 RAG（检索增强生成）

- 添加 Document Loaders（PDF、Word 等）
- 使用 Vector Store（Pinecone、Chroma）
- 实现文档问答功能

3. 优化搜索结果展示

- 在消息中显示搜索来源卡片
- 添加"查看原文"链接
- 高亮引用的搜索结果

❓ 常见问题

Q1: 如何切换搜索 API?

修改 `lib/langchain/tools.js` 中的 API 地址和认证方式。

Q2: 如何限制搜索结果数量?

修改 `bochaWebSearchTool` 的 `count` 参数默认值。

Q3: 如何添加搜索历史记录?

在数据库中添加 `search_logs` 表，记录搜索关键词和结果。

📞 技术支持

如有问题，请检查：

1. ✅ Bocha API Key 是否正确配置
2. ✅ LangChain 依赖包是否安装
3. ✅ 控制台是否有错误日志
4. ✅ 网络连接是否正常