

# vivo BlueLM OpenAI-Compatible API Server

一个功能完整的基于 vivo BlueLM 大模型的 OpenAI 兼容 API 服务器，专为购物反诈场景设计，集成多模态处理、RAG 检索增强、智能 Web 搜索、流式输出、会话管理等企业级功能。

license MIT python 3.8+ FastAPI 0.104.1+ status production ready

## 目录

- 🚀 核心特性
- 💡 应用场景
- 📋 系统要求
- 🛠 快速开始
- 📖 完整 API 文档
- 堅 系统架构
- 🔧 高级配置
- 🛡 安全与最佳实践
- 🚀 生产环境部署
- 📊 监控与运维
- 🔧 故障排除
- 🤝 贡献指南

## 🚀 核心特性

### ⌚ 专业反诈能力

- 智能风险评估**: 基于丰富的反诈知识库，自动识别虚假购物、投资理财、冒充公检法等多种诈骗类型
- 多维度风险分析**: 从价格合理性、平台可信度、付款方式、商品描述等多个维度进行综合评估
- 星级风险评分**: 提供 0-10 星的直观风险评分系统，帮助用户快速判断风险等级

### 🌐 完整 OpenAI 兼容

- 标准 API 格式**: 完全兼容 OpenAI GPT API 规范，支持无缝迁移
- 多模型支持**: 支持 vivo-BlueLM-TB-Pro 和 vivo-BlueLM-V-2.0 多个模型
- 流式响应支持**: 完整实现 Server-Sent Events (SSE) 流式响应，支持实时对话体验

### ImageContext 先进多模态处理

- 智能 OCR 提取**: 高精度图片文字识别，支持多种图片格式
- 深度图片理解**: 详细分析图片内容，包括场景、物体、文字、风格等
- 多格式支持**: 兼容 base64、URL 等多种图片输入格式
- OpenAI Vision 兼容**: 完全支持 OpenAI Vision API 格式

### 💬 RAG 检索增强生成

- 专业知识库**: 基于数千条反诈案例构建的向量知识库
- 语义检索**: 使用 m3e-base 模型进行高质量语义相似度匹配

- **动态上下文**: 实时检索相关知识，增强模型回答的准确性和专业性
- **可控检索**: 用户可自主选择是否启用 RAG 和检索数量

## 🌐 智能 Web 搜索

- **多搜索引擎**: 集成标准搜索、搜狗、夸克、必应等多个搜索引擎
- **智能摘要**: 自动压缩长搜索结果，保留核心信息
- **实时信息**: 获取最新的产品价格、平台评价等实时信息
- **自动调用**: 智能判断是否需要联网搜索，无需手动指定

## ⌚ 高级会话管理

- **多用户隔离**: 支持多用户并发，会话数据完全隔离
- **历史记录**: 智能管理对话历史，支持上下文连续对话
- **用户画像**: 根据用户类型（学生、老师、开发者等）提供个性化服务
- **流式历史记录**: 流式输出的内容也会正确保存到会话历史中

## ⚡ 性能优化

- **异步处理**: 基于 FastAPI 的全异步架构，高并发性能
- **智能缓存**: RAG 检索结果和嵌入向量缓存
- **资源管理**: 自动管理会话历史长度，防止内存溢出
- **错误恢复**: 完善的错误处理和降级机制

## 💡 应用场景

### 💻 消费者保护

- **购物咨询**: 分析商品价格是否合理，识别低价诱骗
- **平台验证**: 评估电商平台、社交平台的可信度
- **支付安全**: 识别不安全的支付方式和转账要求
- **图片分析**: 分析商品图片、聊天截图中的可疑信息

### 风控

- **员工培训**: 为企业员工提供反诈意识培训
- **风险预警**: 实时监测和预警潜在的诈骗风险
- **合规检查**: 协助企业进行交易合规性检查
- **客服集成**: 作为智能客服的反风控模块

### 🎓 教育科研

- **反诈教育**: 为学校和教育机构提供反诈教育工具
- **案例研究**: 支持反诈相关的学术研究和案例分析
- **数据分析**: 提供诈骗趋势分析和统计数据
- **教学辅助**: 辅助反诈相关课程的教学工作

## 📋 系统要求

### 基础环境

- **Python:** 3.8+ (推荐 3.9+)
- **操作系统:** Windows 10+, Ubuntu 18.04+, macOS 10.15+
- **内存:** 最低 4GB, 推荐 8GB+
- **存储:** 最低 2GB 可用空间

## Python 依赖

```
fastapi>=0.104.1
uvicorn[standard]>=0.24.0
pydantic>=2.0.0
numpy>=1.21.0
requests>=2.28.0
python-dotenv>=1.0.0
```

## API 依赖

- **vivo AI 平台账户:** 需要有效的 APP\_ID 和 APP\_KEY
- **Web 搜索服务:** 智谱清言 Web Search API (可选)

## 🚀 快速开始

### 1. 环境准备

```
# 克隆项目
git clone https://github.com/your-org/vivo-bluelm-server.git
cd vivo-bluelm-server

# 创建虚拟环境 (推荐)
python -m venv venv
source venv/bin/activate # Linux/Mac
# venv\Scripts\activate # Windows

# 安装依赖
pip install -r requirements.txt
```

### 2. 配置设置

复制环境变量模板并配置：

```
cp .env.example .env
```

编辑 `.env` 文件：

```
# vivo AI 平台配置
VIVO_APP_ID=your_app_id_here
```

```
VIVO_APP_KEY=your_app_key_here

# API 端点配置
VIVO_GPT_API_URI=/vivogpt/completions
VIVO_GPT_API_DOMAIN=api-ai.vivo.com.cn
MULTIMODAL_URI=/vivogpt/completions
MULTIMODAL_DOMAIN=api-ai.vivo.com.cn
RAG_API_URI=/embedding-model-api/predict/batch
RAG_API_DOMAIN=api-ai.vivo.com.cn

# Web 搜索配置 (可选)
WEB_SEARCH_API_KEY=your_web_search_key
WEB_SEARCH_URL=https://open.bigmodel.cn/api/paas/v4/web_search
```

### 3. 知识库准备

确保知识库文件存在：

```
# 检查知识库文件
ls -la knowledge_base_embeddings/all_knowledge_embeddings.json

# 如果文件不存在, 请联系项目维护者获取
```

### 4. 启动服务

```
# 开发环境启动
python newserver.py

# 或使用 uvicorn (推荐生产环境)
uvicorn newserver:app --host 0.0.0.0 --port 8000 --reload
```

### 5. 验证安装

```
# 检查服务状态
curl http://localhost:8000/v1/health

# 检查可用模型
curl http://localhost:8000/v1/models
```

## 完整 API 文档

### 基础信息

- **Base URL:** <http://localhost:8000>
- **API 版本:** v1

- **认证方式:** Bearer Token (可选, 用于访问控制)
- **内容类型:** application/json

## 完整端点列表

### 1. 模型管理

#### 列出可用模型

```
GET /v1/models
```

响应格式:

```
{
  "object": "list",
  "data": [
    {
      "id": "vivo-BlueLM-TB-Pro",
      "object": "model",
      "created": 1703025600,
      "owned_by": "vivo"
    },
    {
      "id": "vivo-BlueLM-V-2.0",
      "object": "model",
      "created": 1703025600,
      "owned_by": "vivo"
    }
  ]
}
```

### 2. 聊天补全 (核心功能)

#### 创建聊天补全

```
POST /v1/chat/completions
```

基础请求参数:

参数	类型	必需	默认值	说明
model	string	<input checked="" type="checkbox"/>	-	使用的模型名称
messages	array	<input checked="" type="checkbox"/>	-	对话消息列表
temperature	float	<input checked="" type="checkbox"/>	0.7	控制输出随机性 (0.0-2.0)

参数	类型	必需	默认值	说明
max_tokens	integer	✗	1024	最大生成 token 数
top_p	float	✗	1.0	核采样参数 (0.0-1.0)
stream	boolean	✗	false	是否启用流式响应
user	string	✗	-	用户标识符
enable_rag	boolean	✗	true	是否启用 RAG 检索
rag_top_k	integer	✗	2	RAG 检索返回条数
extra	object	✗	{}	额外的模型参数

### 消息格式支持:

#### 1. 纯文本消息:

```
{
  "role": "user",
  "content": "这个商品价格合理吗？"
}
```

#### 2. OpenAI Vision 格式:

```
{
  "role": "user",
  "content": [
    {
      "type": "text",
      "text": "分析这个商品页面"
    },
    {
      "type": "image_url",
      "image_url": {
        "url": "data:image/jpeg;base64,/9j/4AAQ..."
      }
    }
  ]
}
```

#### 3. Base64 图片格式:

```
{
  "role": "user",
  "content": "data:image/jpeg;base64,/9j/4AAQ..."
}
```

## 流式响应格式：

启用 `stream: true` 时，响应采用 Server-Sent Events 格式：

```
data: {"id":"chatcmpl-xxx","object":"chat.completion.chunk","created":1703025600,"model":"vivo-BlueLM-TB-Pro","choices":[{"index":0,"delta":{"role":"assistant"},"finish_reason":null}]}

data: {"id":"chatcmpl-xxx","object":"chat.completion.chunk","created":1703025600,"model":"vivo-BlueLM-TB-Pro","choices":[{"index":0,"delta":{"content":"我帮"},"finish_reason":null}]}

data: {"id":"chatcmpl-xxx","object":"chat.completion.chunk","created":1703025600,"model":"vivo-BlueLM-TB-Pro","choices":[{"index":0,"delta":{"content":"你看"},"finish_reason":null}]}

data: [DONE]
```

## 3. 系统监控

### 健康检查

```
GET /v1/health
```

响应内容：

```
{
  "status": "healthy",
  "timestamp": 1703025600,
  "rag_available": true,
  "active_sessions": 12,
  "system_info": {
    "rag_initialized": true,
    "knowledge_base_size": 10297
  }
}
```

### 服务器统计

```
GET /v1/stats
```

响应内容：

```
{  
    "active_sessions": 12,  
    "total_messages": 1847,  
    "rag_status": "available",  
    "knowledge_base_entries": 10297  
}
```

## 📋 根路径信息

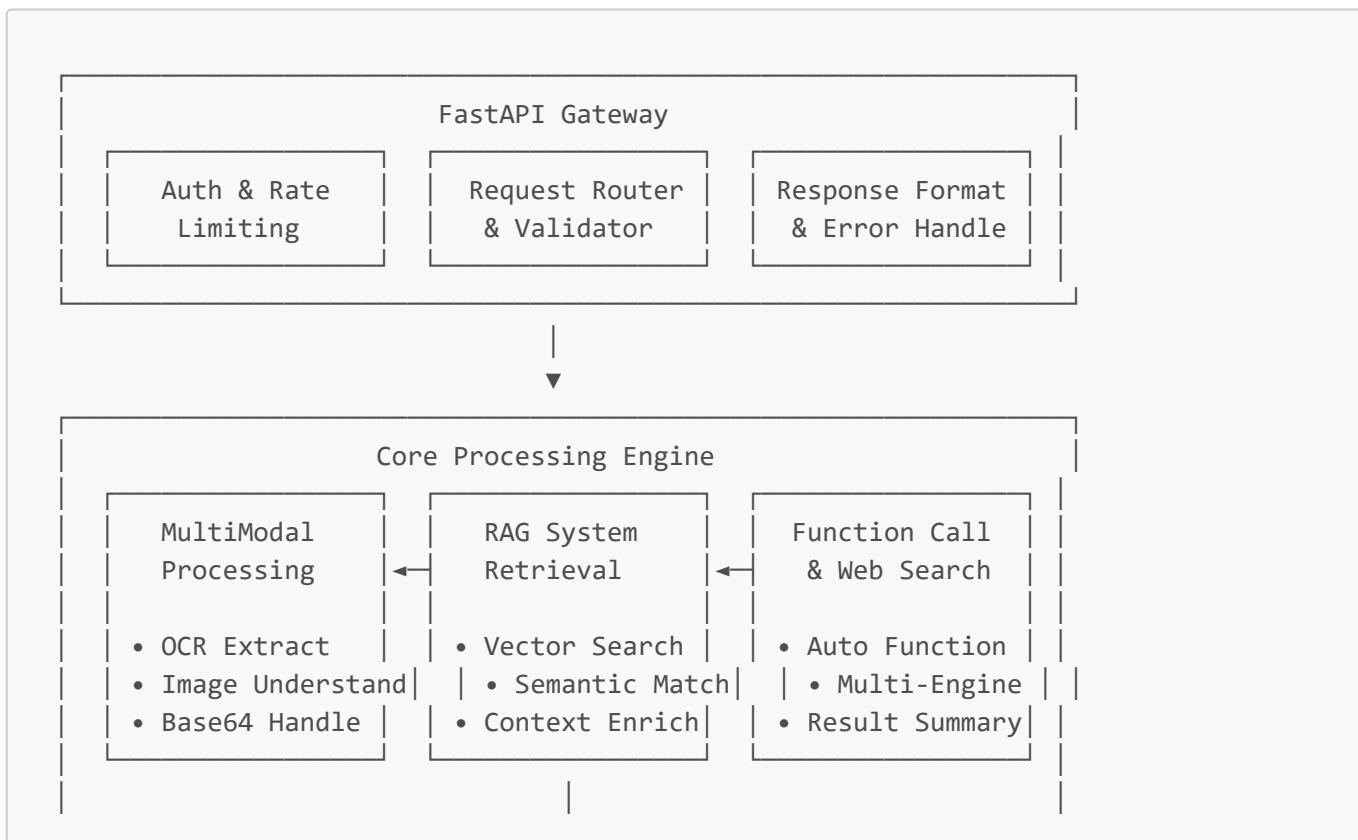
```
GET /
```

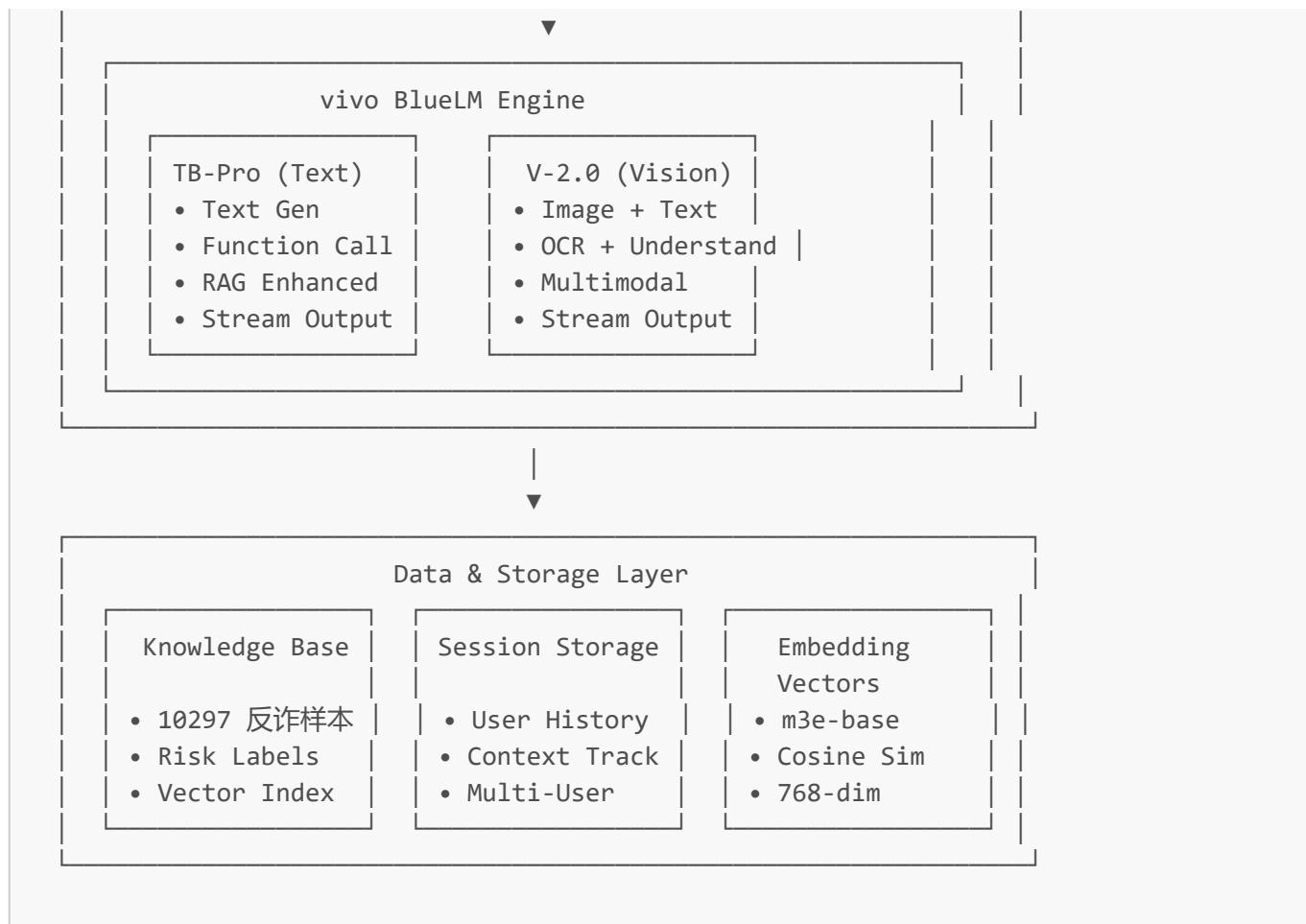
## 响应内容:

```
{  
    "message": "OpenAI-Compatible Server for vivo BlueLM",  
    "version": "1.0.0",  
    "endpoints": ["/v1/models", "/v1/chat/completions"],  
    "features": ["RAG", "MultiModal", "WebSearch", "ConversationHistory",  
    "StreamingResponse"]  
}
```

# █ 系统架构

## ❖ 核心组件架构





## ⌚ 请求处理流程

1. **请求接收**: FastAPI 接收并验证请求格式
2. **消息解析**: 支持文本、多模态、OpenAI Vision 等多种格式
3. **多模态处理**: OCR 文字提取 + 图片内容理解
4. **购物相关性判断**: 自动识别是否为购物相关咨询
5. **RAG 检索**: 根据用户查询检索相关反诈知识
6. **第一次 LLM 调用**: 判断是否需要工具调用
7. **Web 搜索**: 根据需要进行联网搜索
8. **搜索结果处理**: 智能摘要压缩长结果
9. **第二次 LLM 调用**: 生成最终回复
10. **响应格式化**: 标准化 OpenAI 格式输出
11. **会话历史更新**: 保存对话记录

## 📦 模块详细说明

### 1. newserver.py - 核心服务器

- FastAPI 应用入口和路由管理
- OpenAI 兼容 API 实现
- 流式响应处理
- 多模态消息格式转换
- 会话历史管理
- 错误处理和响应格式化

## 2. **MultiModal.py** - 多模态处理引擎

- 高精度 OCR 文字提取
- 深度图片内容理解
- Base64 图片数据处理
- 支持购物场景特定的图片分析

## 3. **vivogpt.py** - LLM 引擎核心

- 同步和流式 LLM 调用
- vivo BlueLM API 封装
- 请求签名和身份认证
- 系统消息处理

## 4. **rag.py** - RAG 检索增强系统

- 向量嵌入生成和管理
- 余弦相似度计算
- 知识库检索和匹配
- 上下文增强处理

## 5. **function\_call.py** - 工具调用管理

- Function Call 解析
- Web 搜索 API 集成
- 多搜索引擎支持
- 搜索结果处理

## 6. **prompt.py** - 提示词工程

- 购物反诈专业提示词
- 用户类型适配
- 风险评分体系
- 对话风格管理

## 7. **schemas.py** - 数据模型定义

- Pydantic 数据验证
- OpenAI 兼容数据结构
- 类型安全保证
- 请求响应模型

## 8. **auth\_util.py** - 认证工具

- vivo AI 平台签名认证
- HMAC-SHA256 签名生成
- 请求头构造

## 🔑 高级配置

### 🌐 环境变量配置

```
# =====
#           vivo AI 平台核心配置
# =====
VIVO_APP_ID=your_app_id_here
VIVO_APP_KEY=your_app_key_here

# =====
#           API 服务端点配置
# =====
VIVO_GPT_API_URI=/vivogpt/completions
VIVO_GPT_API_DOMAIN=api-ai.vivo.com.cn
MULTIMODAL_URI=/vivogpt/completions
MULTIMODAL_DOMAIN=api-ai.vivo.com.cn
RAG_API_URI=/embedding-model-api/predict/batch
RAG_API_DOMAIN=api-ai.vivo.com.cn

# =====
#           Web 搜索服务配置 (可选)
# =====
WEB_SEARCH_API_KEY=your_search_api_key
WEB_SEARCH_URL=https://open.bigmodel.cn/api/paas/v4/web_search

# =====
#           服务器运行配置
# =====
SERVER_HOST=0.0.0.0
SERVER_PORT=8000
DEBUG_MODE=false
LOG_LEVEL=INFO

# =====
#           性能优化配置
# =====
MAX_CONCURRENT_REQUESTS=100
REQUEST_TIMEOUT_SECONDS=30
RAG_CACHE_TTL_SECONDS=3600
CONVERSATION_HISTORY_LIMIT=100
```

### ⚙️ 流式响应配置

```
# 流式响应相关配置
STREAM_CONFIG = {
    "chunk_size": 1024,          # 流式块大小
    "timeout_seconds": 100,       # 流式超时时间
    "retry_attempts": 3,         # 重试次数
```

```
        "buffer_size": 8192          # 缓冲区大小
    }
```

## ⌚ RAG 系统配置

```
# RAG 检索配置
RAG_CONFIG = {
    "top_k": 2,                  # 默认检索数量
    "similarity_threshold": 0.0,  # 相似度阈值
    "max_context_length": 2000,   # 最大上下文长度
    "enable_rerank": False,      # 是否启用重排序
    "embedding_model": "m3e-base", # 嵌入模型
    "vector_dim": 768            # 向量维度
}
```

## 🔍 搜索引擎配置

```
# 搜索引擎配置
SEARCH_CONFIG = {
    "default_engine": "search_std",
    "available_engines": [
        "search_std",
        "search_pro",
        "search_pro_sogou",
        "search_pro_quark",
        "search_pro_jina",
        "search_pro_bing"
    ],
    "default_count": 4,
    "content_size": "medium",
    "timeout_seconds": 10
}
```

# ⌚ 安全与最佳实践

## 🔒 安全配置

### 1. API 密钥管理:

- 使用环境变量存储敏感信息
- 定期轮换 API 密钥
- 不在代码中硬编码密钥

### 2. 访问控制:

- 配置适当的 CORS 策略
- 实施速率限制

- 添加 API 密钥认证 (可选)

### 3. 数据安全:

- 用户会话数据隔离
- 敏感信息过滤
- 定期清理历史记录

## ◆ 性能优化

### 1. 并发处理:

- 使用异步编程模式
- 合理设置并发限制
- 实施请求队列管理

### 2. 缓存策略:

- RAG 检索结果缓存
- 嵌入向量缓存
- 搜索结果缓存

### 3. 资源管理:

- 自动清理过期会话
- 限制历史记录长度
- 监控内存使用

## 🚀 生产环境部署

### dock Docker 部署

```
FROM python:3.9-slim

WORKDIR /app
COPY requirements.txt .
RUN pip install -r requirements.txt

COPY ..
EXPOSE 8000

CMD ["uvicorn", "newserver:app", "--host", "0.0.0.0", "--port", "8000"]
```

### 🔧 Systemd 服务配置

```
[Unit]
Description=vivo BlueLM API Server
After=network.target
```

```
[Service]
Type=simple
User=www-data
WorkingDirectory=/opt/vivo-bluelm-server
Environment=PATH=/opt/vivo-bluelm-server/venv/bin
ExecStart=/opt/vivo-bluelm-server/venv/bin/uvicorn newserver:app --host 0.0.0.0 --port 8000
Restart=always

[Install]
WantedBy=multi-user.target
```

## ④ 反向代理配置 (Nginx)

```
upstream bluelm_backend {
    server 127.0.0.1:8000;
}

server {
    listen 80;
    server_name your-domain.com;

    location / {
        proxy_pass http://bluelm_backend;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;

        # 流式响应支持
        proxy_buffering off;
        proxy_cache off;
        proxy_read_timeout 300s;
    }
}
```

## ■ 监控与运维

### ■ 关键指标监控

#### 1. 服务健康:

- API 响应时间
- 错误率统计
- 并发连接数

#### 2. 业务指标:

- 活跃会话数量
- RAG 检索成功率

- 搜索调用频率

### 3. 资源使用:

- CPU 和内存使用率
- 磁盘空间使用
- 网络带宽使用

## 日志管理

```
# 日志配置建议
LOGGING_CONFIG = {
    "version": 1,
    "formatters": {
        "default": {
            "format": "%(asctime)s - %(name)s - %(levelname)s - %(message)s",
        },
    },
    "handlers": {
        "default": {
            "formatter": "default",
            "class": "logging.StreamHandler",
            "stream": "ext://sys.stdout",
        },
        "file": {
            "formatter": "default",
            "class": "logging.FileHandler",
            "filename": "app.log",
        },
    },
    "root": {
        "level": "INFO",
        "handlers": ["default", "file"],
    },
}
```

## 故障排除

### 常见问题解决

#### 1. 服务启动失败:

- 检查端口是否被占用
- 验证环境变量配置
- 查看详细错误日志

#### 2. API 调用失败:

- 验证 vivo AI 平台凭证
- 检查网络连接
- 确认 API 配额

### 3. RAG 系统不工作:

- 检查知识库文件是否存在
- 验证向量维度是否匹配
- 确认嵌入服务可用性

### 4. 流式响应问题:

- 检查网络连接稳定性
- 确认客户端支持 SSE
- 查看流式响应日志

### 5. 多模态处理失败:

- 验证图片格式是否支持
- 检查 base64 编码是否正确
- 确认多模态模型可用性

## 性能调优建议

### 1. 提高响应速度:

- 调整模型参数降低延迟
- 优化 RAG 检索算法
- 使用缓存减少重复计算

### 2. 增强稳定性:

- 实施熔断机制
- 添加重试逻辑
- 设置合理的超时时间

### 3. 扩展性优化:

- 使用负载均衡
- 实施水平扩展
- 优化数据存储

## 🤝 贡献指南

### 开发环境设置

1. Fork 本项目
2. 创建功能分支: `git checkout -b feature/amazing-feature`
3. 提交更改: `git commit -m 'Add amazing feature'`
4. 推送分支: `git push origin feature/amazing-feature`
5. 提交 Pull Request

### 代码规范

- 遵循 PEP 8 Python 代码规范
- 使用类型注解提高代码可读性

- 添加适当的文档字符串
- 编写单元测试覆盖新功能

## 问题反馈

如遇到问题或有改进建议，请通过以下方式联系：

- 提交 GitHub Issue
- 发送邮件到项目维护者
- 参与社区讨论

---

## 📄 许可证

本项目采用 MIT 许可证。详见 [LICENSE](#) 文件。

## 🙏 致谢

感谢以下项目和团队的支持：

- vivo AI 开放平台
- FastAPI
- Pydantic
- 所有贡献者和用户

---

**版本:** 1.0.0

**最后更新:** 2025年6月20日

**维护者:** xxx 团队