Relatório - Sistema de Suporte à Pesquisa em Cafeicultura Baseado em Inteligência Artificial

GCC 129 - Sistemas Distribuídos

Caio César da Rocha
Gustavo Henrique Moraes Filho
João Victor Vieira Neto Matos
Mateus Henrique Teixeira

Universidade Federal de Lavras
29 de outubro de 2025

Sumário

1 Introdução

A cafeicultura é um dos pilares do agronegócio brasileiro, com o Sul de Minas Gerais se destacando como a região de maior expressão e centro geográfico da cultura no país. Neste cenário de alta competitividade, a pesquisa científica, a inovação e a rápida difusão de tecnologias são fundamentais para garantir a sustentabilidade e a produtividade da lavoura.

Ciente deste desafio, a Universidade Federal de Lavras (UFLA) estabeleceu, em 1995, o Núcleo de Estudos em Cafeicultura (NECAF), hoje parte integrante da Agência de Inovação do Café (INOVACAFÉ). O NECAF tem como missão primordial ser um difusor de tecnologia, "estreitando o laço existente entre a Universidade e o produtor rural", conforme sua visão fundadora. Este núcleo é composto por um corpo técnico-científico robusto de professores, pesquisadores, alunos de pós-graduação e graduação.

Figura 1: Logo do núcleo de estudos NECAF

Esta atividade de pesquisa e extensão gera um volume substancial e crescente de conhecimento técnico e científico, documentado em artigos, publicações em congressos, relatórios de projetos e manuais técnicos. No entanto, o acesso eficiente e contextualizado a essa vasta base de dados representa um desafio operacional significativo. Pesquisadores e alunos frequentemente necessitam de respostas rápidas para questões específicas, e a consulta manual a múltiplos documentos pode ser um processo lento e fragmentado, dificultando a tomada de decisão e o fluxo da pesquisa.

Visando solucionar esta lacuna de acesso à informação, o presente trabalho propõe o desenvolvimento de um sistema de assistente virtual inteligente. A solução utiliza uma arquitetura moderna para fornecer aos membros do NECAF um ponto de acesso centralizado e conversacional ao acervo de conhecimento do núcleo. O sistema é acessado via **Telegram**, permitindo consultas por texto ou áudio. Mensagens de áudio são processadas por um serviço de transcrição (**API OpenAI**), enquanto as consultas textuais são direcionadas a um modelo de linguagem de grande escala (LLM) da plataforma **Ollama**, executado localmente. Este modelo é potencializado pela técnica de Geração Aumentada por Recuperação (RAG), que o capacita a consultar a base de conhecimento privada do NECAF para formular respostas precisas e contextuais.

1.1 Objetivos

O objetivo geral deste trabalho é projetar e implementar um sistema de inteligência artificial conversacional para otimizar o acesso à informação técnica e científica sobre cafeicultura para os membros do Núcleo de Estudos em Cafeicultura (NECAF).

Para alcançar este objetivo, foram definidos os seguintes objetivos específicos:

- Modelar uma arquitetura de sistema que integre o Telegram como interface de usuário, permitindo consultas por texto e áudio;

- Implementar um serviço de transcrição de áudio para converter as perguntas faladas em formato textual;
- Configurar e hospedar um modelo de linguagem de grande escala (Ollama) localmente, garantindo a privacidade e o controle do processamento dos dados;
- Desenvolver e alimentar um mecanismo de Geração Aumentada por Recuperação (RAG) com artigos e documentos técnicos relevantes do NECAF;
- Integrar os componentes em um fluxo operacional coeso, onde as consultas são processadas e as respostas são geradas com base nos documentos do núcleo;
- Validar a eficácia do sistema em fornecer respostas rápidas e precisas às consultas dos pesquisadores.

2 Arquitetura do Sistema

A arquitetura do sistema foi projetada para ser modular, resiliente e escalável, integrando de forma coesa serviços de nuvem (Cloud) e serviços locais (On-Premise). O design da solução combina dois padrões arquiteturais principais, que são fundamentais para seu funcionamento: a Arquitetura Orientada a Serviços (SOA) e a Arquitetura Orientada a Eventos (EDA).

A Figura 2 ilustra o diagrama de fluxo de dados completo da aplicação, desde a entrada da consulta do usuário no Telegram até a geração da resposta final pela inteligência artificial local.

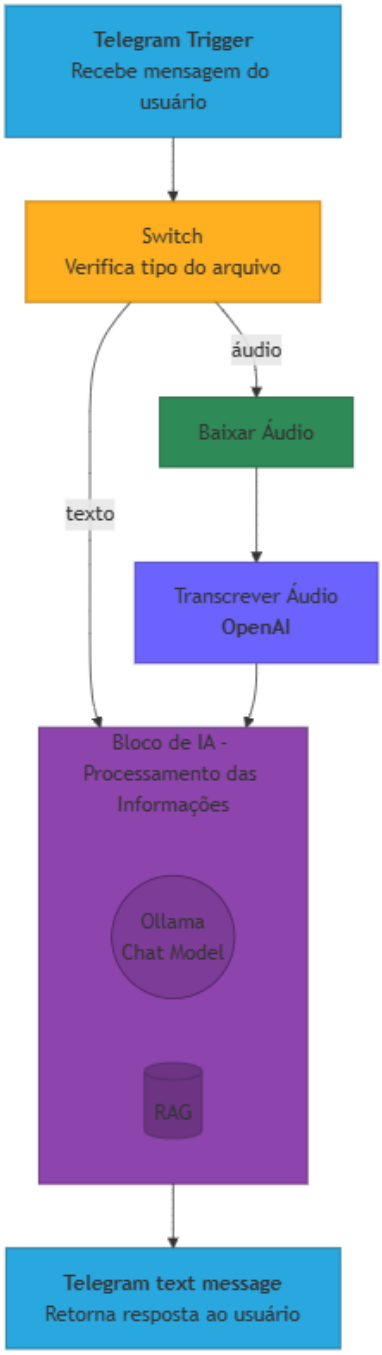

Figura 2: Diagrama do fluxo de dados da aplicação, da entrada no Telegram à resposta do OLLAMA.

2.1 Arquitetura Orientada a Serviços (SOA)

A Arquitetura Orientada a Serviços é um paradigma no qual a aplicação é composta por um conjunto de serviços independentes e fracamente acoplados, que se comunicam por meio de interfaces bem definidas (geralmente APIs).

Este padrão está presente no projeto, pois o sistema não é um bloco único (monólito), mas sim um **orquestrador** de múltiplos serviços distintos, cada um com uma responsabilidade única. Esta abordagem adota um modelo de **arquitetura híbrida**, combinando:

- **Serviços Externos (Cloud):** O sistema consome APIs de terceiros para tarefas específicas. São eles:
  - **API do Telegram:** Atua como a interface de usuário (frontend) para recebimento e envio de mensagens.
  - **API da OpenAI (ChatGPT):** É consumida como um serviço sob demanda especificamente para a transcrição de áudio em texto.
- **Serviços Internos (On-Premise):** O núcleo de inteligência e os dados sensíveis do NECAF são executados localmente, garantindo a soberania e a privacidade dos dados.
  - **Serviço de Inferência (OLLAMA):** O container que executa o modelo de linguagem de grande escala (LLM).
  - **Base RAG (Vetorização):** A base de conhecimento (artigos) que o OLLAMA consulta.

A vantagem desta arquitetura é a modularidade. Se, por exemplo, o serviço de transcrição da OpenAI se tornar inviável, ele pode ser substituído por outro (como um modelo Whisper local) sem a necessidade de alterar o restante do fluxo, apenas o nó de serviço correspondente.

Figura 3: Princípios da Arquitetura Orientada a Serviços

2.2 Arquitetura Orientada a Eventos (EDA)

A Arquitetura Orientada a Eventos (EDA) é um padrão no qual o fluxo do sistema é determinado pela ocorrência de "eventos"(como ações do usuário ou mensagens de outros sistemas). A aplicação permanece reativa, "ouvindo"por eventos e disparando ações em resposta.

Este padrão é a espinha dorsal de como o sistema opera:

1. **O Evento:** O sistema permanece inativo até que um evento ocorra. Neste caso, o evento é um usuário enviando uma nova mensagem (texto ou áudio) para o bot.
2. **O Gatilho (Trigger):** O componente **"Telegram Trigger1"** (visível na Figura 2) atua como um *listener* (ouvinte) de eventos, configurado para "ouvir"o evento ‘Updates: message‘.
3. **A Reação (Fluxo):** A captura desse evento dispara toda a cadeia de processos subsequentes de forma assíncrona: o roteamento pelo "Switch", a chamada condicional aos serviços de transcrição ou inferência, e o envio da resposta.

O uso de EDA é ideal para aplicações de chatbot, pois o fluxo é inerentemente assíncrono e reativo à entrada do usuário, permitindo que o sistema lide com múltiplas solicitações simultâneas de forma eficiente sem manter uma conexão persistente.

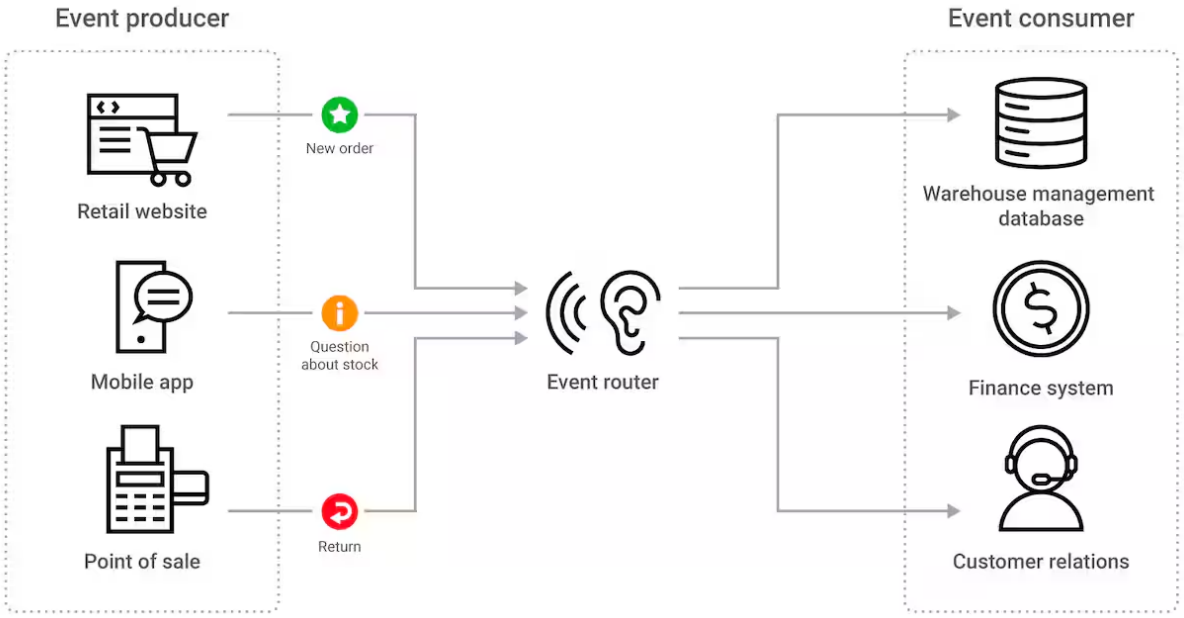

Figura 4: Princípios da Arquitetura Orientada a Eventos

3 Modelagem de Ameaças

Para garantir a segurança e a confiabilidade do sistema proposto, foi realizada uma análise de riscos utilizando o modelo **STRIDE**. Esta metodologia, padrão na indústria, classifica as ameaças em seis categorias (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privilege) e permite identificar proativamente as vulnerabilidades da arquitetura.

A Tabela 1 apresenta um resumo de alto nível das categorias de ameaças aplicadas ao contexto da solução. Uma análise detalhada de cada ameaça específica, juntamente com as contramedidas e mitigações propostas, é apresentada nas subseções seguintes.

Tabela 1: Resumo da Modelagem de Ameaças STRIDE

Categoria STRIDE	**Descrição**	**Ameaça de Alto Nível no Sistema**
Spoofing (Falsificação)	Fazer-se passar por outro usuário ou componente.	Acesso não autorizado ao bot por usuários externos ao NECAF ou falsificação de requisições de API ao backend.
Tampering (Adulteração)	Modificar dados em trânsito ou em repouso.	Adulteração da base de conhecimento (RAG), manipulação da IA (Prompt Injection) ou interceptação de dados em trânsito.
Repudiation (Repúdio)	Negar ter realizado uma ação.	Um usuário negar ter enviado uma consulta específica ou alegar que a resposta recebida foi alterada.
Information Disclosure (Divulgação)	Exposição de dados sensíveis a partes não autorizadas.	Vazamento de dados de áudio (para a OpenAI), de chaves de API, ou do conteúdo das perguntas e artigos do NECAF.
Denial of Service (Negação de Serviço)	Tornar o sistema indisponível.	Esgotamento de recursos da IA local (OLLAMA) por sobrecarga, falha nas APIs externas ou envio massivo de mensagens.
Elevation of Privilege (Elevação)	Obter permissões de acesso que não deveria ter.	Execução de código malicioso no container OLLAMA ou uso do container como pivô para acesso indevido à rede interna.

3.1 Análise Detalhada e Mitigações

A seguir, cada categoria de ameaça é detalhada, apresentando os vetores de ataque específicos e as contramedidas implementadas ou recomendadas.

3.1.1 Spoofing (Falsificação)

Esta ameaça envolve um atacante fingindo ser uma entidade legítima.

Acesso de Usuário Não Autorizado Uma pessoa não afiliada ao NECAF poderia descobrir o bot e começar a usá-lo, consumindo recursos e acessando dados.

- **Mitigação:** O backend deve implementar uma "allow list"(lista de permissão), verificando o ‘userid‘ do Telegram de cada requisição contra uma lista pré-aprovada de membros do NECAF. Mensagens de usuários não autorizados devem ser descartadas.

Falsificação de Requisições de API Um atacante poderia descobrir o endpoint do webhook do Telegram e enviar requisições falsas, sobrecarregando o sistema ou injetando dados.

- **Mitigação:** Utilizar o ‘secrettoken‘ fornecido pelo Telegram na configuração do webhook. O backend deve validar este token a cada requisição para garantir que ela se origina do Telegram.

3.1.2 Tampering (Adulteração)

Esta ameaça envolve a modificação não autorizada de dados.

Envenenamento de Dados (RAG) Um atacante com acesso ao servidor poderia modificar os artigos da base de conhecimento (RAG). Isso faria com que a IA gerasse respostas factualmente incorretas, tendenciosas ou maliciosas.

- **Mitigação:** Aplicar controle de integridade (ex: armazenar hashes dos arquivos) e implementar permissões de sistema de arquivos rigorosas. A conta de serviço que executa o OLLAMA deve ter acesso somente-leitura (*read-only*) ao repositório dos artigos.

Injeção de Prompt (Prompt Injection) Um usuário (malicioso ou curioso) poderia enviar um texto ou áudio com instruções para "enganar"a IA, fazendo-a ignorar suas diretrizes originais (o RAG) ou responder sobre tópicos inadequados.

- **Mitigação:** Implementar "system prompts"robustos (instruções claras para a IA) e realizar a sanitização das entradas do usuário, filtrando metacaracteres ou instruções que conflitem com o objetivo do sistema.

3.1.3 Repudiation (Repúdio)

Esta ameaça consiste na negação de ter realizado uma ação.

Repúdio de Consulta Um usuário poderia alegar que não fez uma pergunta específica ou que a resposta fornecida pelo sistema foi alterada após o recebimento.

- **Mitigação:** Implementar logs de auditoria seguros e imutáveis (apenas *append*). Os logs devem registrar o 'user$_id$'$doTelegram, o*timestamp*, aconsultarecebidaearespostagerada$

3.1.4 Information Disclosure (Divulgação de Informações)

Esta é uma das ameaças mais críticas, envolvendo a exposição de dados sensíveis.

Vazamento de Dados de Áudio (Risco Alto) Os áudios enviados pelos pesquisadores podem conter ideias de pesquisa inéditas ou dados sensíveis. Estes áudios são enviados para a API da OpenAI (um terceiro) para transcrição.

- **Mitigação:** A mitigação ideal é substituir a API externa por um modelo de *speech-to-text* local (ex: Whisper rodando *on-premise*). Isso mantém todos os dados sensíveis dentro da infraestrutura da UFLA.

Vazamento de Chaves de API As chaves do bot do Telegram e da API da OpenAI, se expostas no código-fonte, em logs ou em variáveis de ambiente inseguras, permitiriam o sequestro dos serviços.

- **Mitigação:** Utilizar um sistema de gerenciamento de segredos (ex: Docker secrets, HashiCorp Vault, ou variáveis de ambiente injetadas no container) e nunca "hard-codar"chaves no código.

Vazamento de Dados de Pesquisa Um atacante que obtenha acesso aos logs, ao container do OLLAMA ou à base RAG poderia roubar as perguntas (o que o NECAF está pesquisando) ou os próprios artigos.

- **Mitigação:** Criptografar a base RAG em repouso, restringir o acesso aos logs e isolar a rede do container OLLAMA.

3.1.5 Denial of Service (Negação de Serviço)

Esta ameaça foca em tornar o sistema indisponível para os usuários legítimos.

Sobrecarga da IA Local (OLLAMA) Consultas complexas e simultâneas (intencionais ou não) podem esgotar os recursos de CPU, GPU ou RAM do container OLLAMA, travando o serviço.

- **Mitigação:** Implementar uma fila de processamento no backend e aplicar *Rate Limiting* (limite de taxa, ex: 5 mensagens por minuto) por usuário.

Falha na API Externa (OpenAI) Se a API da OpenAI ficar indisponível ou atingir o limite de taxa, a funcionalidade de áudio do bot irá falhar.

- **Mitigação:** Implementar tratamento de erros robusto, com *retries* (novas tentativas) e *exponential backoff*, e um *circuit breaker* para parar de enviar requisições. O bot deve informar ao usuário que o serviço de áudio está temporariamente indisponível.

3.1.6 Elevation of Privilege (Elevação de Privilégio)

Esta ameaça ocorre quando um atacante obtém permissões que não deveria ter.

Execução Remota de Código (RCE) via OLLAMA Uma consulta especialmente criada (via Prompt Injection avançado) poderia explorar uma vulnerabilidade no OLLAMA para executar comandos dentro do container.

- **Mitigação:** Aplicar defesa em camadas. Rodar o processo OLLAMA dentro do container com um usuário de privilégios mínimos (*non-root*) e manter a imagem do OLLAMA sempre atualizada com os últimos patches de segurança.

Acesso à Rede Interna Se o container OLLAMA for comprometido, o atacante poderia usá-lo como "pivô"para escanear e atacar outros serviços na rede local da UFLA.

- **Mitigação:** Implementar segmentação de rede. O container do OLLAMA deve estar em uma rede isolada (ex: Docker network) que *apenas* permite a comunicação com o backend e a base RAG, bloqueando todo o resto do tráfego de saída.

3.2 Priorização de Riscos

Com base na análise STRIDE, três riscos destacam-se pelo seu alto impacto e probabilidade no contexto específico do NECAF:

1. **Divulgação de Informações (I):** O envio de áudios de pesquisa para uma API de terceiros é o risco de confidencialidade mais significativo. A troca por um modelo de transcrição local (*on-premise*) deve ser a principal prioridade de segurança.
2. **Adulteração (T) - Envenenamento de Dados:** A integridade da base RAG é sinônimo da confiabilidade do sistema. Se os artigos forem adulterados, o sistema perde sua função. O controle de acesso e a integridade dos arquivos são mandatórios.
3. **Adulteração (T) - Prompt Injection:** A natureza dos LLMs torna a injeção de prompt uma ameaça persistente. O refinamento contínuo das instruções do sistema e da sanitização das entradas é crucial para evitar que a IA seja usada de forma maliciosa.