



**Politécnico
de Viseu**

Escola Superior
de Tecnologia
e Gestão de Viseu

Desenvolvimento de Chatbot Agrícola Inteligente Integrado ao SIGP para Suporte Técnico e Apoio à Decisão

Alexandre Rodrigues Fernandes

Trabalho de Projeto

Licenciatura em

Engenharia Informática

Junho 2026



**Politécnico
de Viseu**

Escola Superior
de Tecnologia
e Gestão de Viseu



Desenvolvimento de Chatbot Agrícola Inteligente Integrado ao SIGP para Suporte Técnico e Apoio à Decisão

Alexandre Rodrigues Fernandes

Realizado na empresa GeoDouro

Relatório de Projeto

Licenciatura em

Engenharia Informática

Trabalho efetuado sob a orientação de
Joana Rita da Silva Fialho (ESTGV)
Telmo Nogueira (GeoDouro)

Junho de 2026

Dedicatória

Aos meus pais, ao meu irmão, à minha namorada e a toda a minha família, bem
como a todos os que me apoiaram ao longo deste percurso.

Agradecimentos

Quero agradecer a todos os que me acompanharam neste caminho, que me deram alento e ânimo para continuar.

À Professora Joana Fialho pela constante disponibilidade, apoio e orientação.

Ao orientador Telmo Nogueira e Fernando Soares pela partilha de conhecimento, paciência e acompanhamento técnico.

À empresa GeoDouro por esta oportunidade.

À minha família e namorada pelo suporte.

E a todos os meus colegas.

Resumo

O presente relatório descreve o trabalho realizado no âmbito do estágio curricular da Licenciatura em Engenharia Informática da Escola Superior de Tecnologia e Gestão de Viseu, com a duração de quinze semanas, realizado na empresa Geodouro Consultoria e Topografia, Lda. O objetivo principal foi criar um *chatbot*, integrado na plataforma de gestão da empresa. Este assistente consegue responder a dúvidas sobre a plataforma SIGP e consultar as bases de dados através de perguntas em linguagem natural. Todo o processamento ocorre de forma local para garantir a privacidade total dos dados.

O documento aborda a tecnologia atual sobre modelos de linguagem e arquiteturas de recuperação de informação. Explica também como a solução foi desenhada e construída, e apresenta os testes que comprovam o seu bom desempenho.

Os resultados provam que a solução funciona bem e é robusta. O sistema fixou-se numa taxa global de acerto de 72% num teste final com 50 perguntas reais do dia a dia da empresa.

PALAVRAS-CHAVE: Chatbot, Recuperação Aumentada por Geração, Modelos de Linguagem de Grande Escala, Agente de Consulta Estruturada, Agricultura de Precisão, Processamento de Linguagem Natural

Abstract

This report describes the work carried out as part of the fifteen-week internship for the Bachelor's Degree in Computer Engineering at the Escola Superior de Tecnologia e Gestão de Viseu, conducted at Geodouro Consultoria e Topografia, Lda. The main objective was to create a chatbot integrated into the company's management platform. This assistant is capable of answering questions about the SIGP platform and querying databases using natural language. All processing occurs locally to ensure complete data privacy.

The document discusses current technology regarding language models and information retrieval architectures. It also explains how the solution was designed and built, and presents the tests that demonstrate its strong performance.

The results prove that the solution works well and is robust. The system achieved an overall accuracy rate of 72% in a final test with 50 real-world questions from the company's daily operations.

Key-words: Chatbot, Retrieval-Augmented Generation, Large Language Models, Structured Query Agent, Precision Agriculture, Natural Language Processing

Índice

ÍNDICE DE FIGURAS	XI
ÍNDICE DE TABELAS	XIII
LISTA DE SIGLAS / ABREVIATURAS	XIV
1 INTRODUÇÃO.....	16
1.1 ENQUADRAMENTO E MOTIVAÇÃO	16
1.2 ENTIDADE DE ACOLHIMENTO	17
1.3 OBJETIVOS E RESULTADOS ESPERADOS	20
1.4 PLANO DE TRABALHO	21
1.5 ESTRUTURA DO RELATÓRIO	22
2 ESTADO DA ARTE	23
2.1 PROCESSAMENTO DE LINGUAGEM NATURAL.....	23
2.1.1 <i>Redes Recorrentes e as suas Limitações</i>	23
2.1.2 <i>A Arquitetura Transformer</i>	24
2.2 LARGE LANGUAGE MODELS.....	25
2.2.1 <i>Parametrização e Quantização Local</i>	25
2.2.2 <i>Limitações dos LLM e a Necessidade de Conhecimento Externo</i>	27
2.2.3 <i>Janela de Contexto e RAG</i>	28
2.3 BASES DE DADOS VETORIAIS E EMBEDDINGS.....	29
2.4 AGENTES CONVERSACIONAIS (CHATBOTS).....	30
2.5 APLICAÇÕES DE IA NA AGRICULTURA.....	31
2.6 LACUNAS E CONTRIBUIÇÃO	31
2.7 RESUMO	33
3 TECNOLOGIAS E FERRAMENTAS UTILIZADAS	34
3.1 BACKEND	34
3.1.1 <i>Python e FastAPI</i>	34
3.1.2 <i>LangChain e LangGraph</i>	35

3.1.3	<i>ChromaDB</i>	36
3.1.4	<i>Ollama</i>	37
3.1.5	<i>Qwen2.5</i>	37
3.1.6	<i>HuggingFace Embeddings</i>	39
3.1.7	<i>Psycopg2</i>	39
3.2	FRONTEND	39
3.3	FERRAMENTAS DE DESENVOLVIMENTO	40
3.3.1	<i>Git/GitHub</i>	40
3.3.2	<i>VSCode</i>	40
3.3.3	<i>Postman</i>	41
3.3.4	<i>LangFuse</i>	41
3.4	RESUMO	42
4	METODOLOGIA E CONCEÇÃO	44
4.1	METODOLOGIA DE TRABALHO	44
4.1.1	<i>Metodologia Ágil</i>	44
4.1.2	<i>Acompanhamento e Validação</i>	45
4.2	REQUISITOS	45
4.2.1	<i>Requisitos Funcionais</i>	46
4.2.2	<i>Requisitos Não Funcionais</i>	47
4.3	USER STORIES	48
4.4	DIAGRAMA DE CASOS DE USO	49
4.5	CONCEÇÃO DA SOLUÇÃO	51
4.5.1	<i>Arquitetura Geral</i>	51
4.5.2	<i>Pipeline RAG</i>	53
4.5.3	<i>SQL Agent</i>	54
4.5.4	<i>Observabilidade</i>	55
4.6	RESUMO	55
5	ATIVIDADES DESENVOLVIDAS	56
5.1	FASE 1: ANÁLISE E PREPARAÇÃO	56
5.2	FASE 2: CONFIGURAÇÃO DO AMBIENTE DE DESENVOLVIMENTO	57
5.2.1	<i>Configuração do Ambiente de Desenvolvimento</i>	57
5.2.2	<i>Pipeline de Ingestão de Documentos</i>	57
5.3	FASE 3: DESENVOLVIMENTO DO CORE	58
5.3.1	<i>Levantamento das Bases de Dados</i>	58
5.3.2	<i>Sistema RAG</i>	59

5.3.3	<i>API FastAPI</i>	60
5.3.4	<i>Router de Classificação</i>	61
5.3.5	<i>SQL Agent</i>	62
5.4	FASE 4: INTERFACE E OBSERVABILIDADE.....	64
5.4.1	<i>Interface Web</i>	64
5.4.2	<i>Observabilidade com Langfuse</i>	67
5.4.3	<i>Otimizações de Desempenho</i>	69
5.4.4	<i>Melhorias Iterativas à Qualidade das Respostas</i>	70
5.4.5	<i>Otimização Avançada do Pipeline SQL</i>	72
5.4.6	<i>Pesquisa Híbrida BM25+MMR e Schema Pruning</i>	73
5.4.7	<i>Demonstrações de Poucas Amostras Dinâmicas</i>	73
5.4.8	<i>Vistas Materializadas</i>	74
5.5	FASE 5: TESTES	74
5.5.1	<i>Testes Unitários ao Router</i>	74
5.5.2	<i>Testes de Integração SQL</i>	75
5.5.3	<i>Testes Funcionais</i>	75
5.5.4	<i>Validação com Golden Set da Geodouro</i>	76
5.5.5	<i>Avaliação Comparativa de Estratégias de Retrieval</i>	79
5.5.6	<i>Avaliação do Vectorstore de Tabelas</i>	83
5.5.7	<i>Testes de Escalabilidade</i>	84
5.6	FASE 6: FINALIZAÇÃO	87
6	CONCLUSÃO	88
6.1	OBJETIVOS ALCANÇADOS	88
6.2	RESULTADOS E CONTRIBUIÇÕES.....	89
6.3	DIFICULDADES ENCONTRADAS.....	90
6.4	APRENDIZAGENS ADQUIRIDAS.....	90
6.5	TRABALHO FUTURO.....	91
	REFERÊNCIAS BIBLIOGRÁFICAS	93
	ANEXOS	99

Índice de Figuras

FIGURA 1 - GEODOURO	18
FIGURA 2 - MÓDULOS DO SIGP	19
FIGURA 3 - GRÁFICO DE GANTT	21
FIGURA 4 - DESEMPENHO DAS VÁRIAS LLM	38
FIGURA 5 - DIAGRAMA DE CASOS DE USO.....	50
FIGURA 6 - ARQUITETURA DO SISTEMA	51
FIGURA 7 - FLUXO DE INTERAÇÃO.....	52
FIGURA 8 - CICLO REACT DO SQL AGENT	54
FIGURA 9 - CONSTRUÇÃO DO RETRIEVER HÍBRIDO	60
FIGURA 10 - FUNÇÃO DE CLASSIFICAÇÃO DO ROUTER	62
FIGURA 11 - DEFINIÇÃO DAS CINCO FERRAMENTAS DISPONÍVEIS AO AGENTE SQL.....	63
FIGURA 12 - INTERFACE WEB DO CHATBOT	65
FIGURA 13 - INTERFACE DA PÁGINA DE ADMINISTRADOR	66
FIGURA 14 - DASHBOARD DO LANGFUSE	67
FIGURA 15 - SCORES DO LANGFUSE, ONDE SE PODE VER A QUALIDADE DAS RESPOSTAS DE ACORDO COM OS UTILIZADORES	68
FIGURA 16 - FUNÇÃO _COMPRESS_HISTORY: SUBSTITUIÇÃO DE TABELAS NO HISTÓRICO PARA FORÇAR NOVA CONSULTA SQL.....	71
FIGURA 17 - DIAGRAMA DO PIPELINE SQL OPTIMIZADO	72
FIGURA 18 - GRÁFICO DE EVOLUÇÃO DA TAXA DE ACERTO NO GOLDEN SET.....	77
FIGURA 19 - ACERTO POR ESTRATÉGIA DE RETRIEVAL	81
FIGURA 20 – LATÊNCIA POR PERGUNTA.....	82
FIGURA 21 - LATÊNCIA VS ACERTO	83
FIGURA 22 - ESCALABILIDADE DO VECTORSTORE CHROMADB: IMPACTO DO NÚMERO DE CHUNKS NA LATÊNCIA DE PESQUISA (ESQUERDA) E NO TEMPO DE INDEXAÇÃO INICIAL (DIREITA).....	85
FIGURA 23 - COMPARAÇÃO DOS TEMPOS DE RESPOSTA DO SISTEMA ENTRE UM CENÁRIO DE BASELINE (1 UTILIZADOR) E UM CENÁRIO DE STRESS (5 UTILIZADORES CONCORRENTES).....	86
FIGURA 24 - EVOLUÇÃO DA LATÊNCIA CUMULATIVA AO LONGO DA EXECUÇÃO DO TESTE DE CARGA COM 5 UTILIZADORES CONCORRENTES.....	86

Índice de Tabelas

TABELA 1 - TECNOLOGIAS SELECIONADAS PARA CADA CAMADA	42
TABELA 2 - REQUISITOS FUNCIONAIS	46
TABELA 3 - REQUISITOS NÃO FUNCIONAIS	47
TABELA 4 - USER STORIES	48
TABELA 5 - RESULTADOS DOS <i>BENCHMARKS</i> AUTOMATIZADOS	76
TABELA 6 - EVOLUÇÃO DA TAXA DE ACERTO NO GOLDEN SET POR ITERAÇÃO DE OPTIMIZAÇÃO	78
TABELA 7 - COMPARAÇÃO DE ESTRATÉGIAS DE RETRIEVAL	80

Lista de siglas / abreviaturas

API - Application Programming Interface

ANN - Approximate Nearest Neighbor

BM25 - Best Match 25

CPU - Central Processing Unit

GPU - Graphics Processing Unit

HyDE - Hypothetical Document Embeddings

IA - Inteligência Artificial

IoT - Internet of Things

LLM - Large Language Model

LSTM - Long Short-Term Memory

MMR - Maximal Marginal Relevance

MoE - Mixture-of-Experts

NDRE - Normalized Difference Red Edge

NDVI - Normalized Difference Vegetation Index

PAC - Política Agrícola Comum

PEPAC - Plano Estratégico da Política Agrícola Comum

PLN - Processamento de Linguagem Natural

RAG - Retrieval-Augmented Generation

REST - Representational State Transfer

RLHF - Reinforcement Learning from Human Feedback

RNN - Recurrent Neural Network

SIGP - Sistema Integrado de Gestão de Propriedades

SQL - Structured Query Language

SSE - Server-Sent Events

TTL - Time to Live

VRAM - Video Random Access Memory

1 Introdução

O presente capítulo introduz o contexto em que o estágio curricular se insere, estabelecendo as bases fundamentais do projeto desenvolvido. Ao longo das próximas secções, é detalhado o enquadramento tecnológico e a motivação que impulsionaram a criação de um *chatbot* aplicado ao setor agrícola. São também definidos os objetivos a atingir, apresentada a entidade de acolhimento (Geodouro) e delineados os resultados práticos esperados no final deste percurso, finalizando com a apresentação da estrutura geral que orienta o resto do documento.

1.1 Enquadramento e Motivação

A Engenharia Informática desempenha um papel central na transformação digital de todos os setores económicos, incluindo o da agricultura. A integração de Inteligência Artificial (IA), de Processamento de Linguagem Natural (PLN) e de sistemas de informação em ambientes agrícolas representa uma área emergente com um potencial significativo de impacto económico e social.

A agricultura de precisão, baseada em tecnologias de geolocalização, deteção remota e análise de dados, tem evoluído rapidamente nas últimas décadas. Plataformas integradas, como o SIGP (Sistema Integrado de Gestão de Propriedades) da Geodouro, consolidam diversas funcionalidades, desde a análise de índices de vegetação obtidos via drone até à gestão da conformidade regulamentar no âmbito da Política Agrícola Comum (PAC) (SIGP, 2026).

Contudo, a crescente complexidade destas plataformas cria desafios significativos ao nível da usabilidade. Os utilizadores finais como agricultores e técnicos agrícolas, enfrentam frequentemente dificuldades em navegar por documentação técnica

extensa e em compreender a complexa terminologia regulamentar. Acresce a isto o desafio de aplicar corretamente funcionalidades críticas de conformidade, como o Caderno de Campo exigido pelo PEPAC 2023-2027 (GPP, 2026) e de interpretar dados técnicos específicos, nomeadamente os índices de vegetação *Normalized Difference Vegetation Index* (NDVI) e *Normalized Difference Red Edge* (NDRE), o que acaba por dificultar a tomada de decisões informadas com base em múltiplas fontes de informação.

Os sistemas de suporte automatizado baseados em *Large Language Models* (LLM) e arquiteturas de *Retrieval-Augmented Generation* (RAG) representam uma solução promissora para colmatar estes desafios. Os *chatbots* conversacionais podem fornecer suporte técnico contínuo, responder a dúvidas com base em documentação específica e auxiliar na tomada de decisões, reduzindo substancialmente a carga de trabalho das equipas de suporte humano.

A motivação pessoal para o desenvolvimento deste projeto advém do interesse em aplicar conhecimentos de Inteligência Artificial, Processamento de Linguagem Natural e Engenharia de Software num contexto real, contribuindo para a digitalização de um setor económico fundamental como o da agricultura. Por fim, a oportunidade de trabalhar com tecnologias emergentes (LLM de execução local, RAG e bases de dados vetoriais) em ambiente empresarial constitui uma experiência de enorme valor no processo de transição entre a formação académica e a integração na vida profissional.

1.2 Entidade de Acolhimento

A Geodouro - Consultoria e Topografia, Lda. (Figura 1) é uma empresa portuguesa fundada no ano 2000, especializada nas áreas da consultoria agrícola, topografia e no desenvolvimento de soluções tecnológicas direcionadas para a agricultura de precisão. A sua atividade atua primordialmente em três vertentes estratégicas. A primeira centra-se na consultoria técnica agrícola, com especial incidência na viticultura e olivicultura. A segunda foca-se na engenharia geográfica, que engloba a

prestação de serviços de topografia, produção de cartografia e cadastro. Por fim, a terceira vertente dedica-se às tecnologias de informação, destacando-se o desenvolvimento e a comercialização de *software* integrado para a gestão agrícola (GeoDouro, 2026).



Figura 1 - GeoDouro

Esta terceira vertente tecnológica é o Sistema Integrado de Gestão de Propriedades (SIGP), a plataforma central da Geodouro, desenvolvida internamente com o intuito de responder às exigentes necessidades da gestão moderna das explorações agrícolas. A plataforma destaca-se por ser uma aplicação *web* baseada numa arquitetura de dados robusta, operando de forma modular para cobrir o ciclo completo da atividade agrícola e agroindustrial, conforme esquematizado na Figura 2. O seu ecossistema divide-se em valências tecnológicas fundamentais, começando pela vertente de conformidade legal e gestão de campo, que assegura a integração nativa com o SIFITO (SIFITO, 2026) e com as diretrizes da PAC/PEPAC (2023-2027) (GPP, 2026), centralizando o registo de todas as práticas, custos e intervenções no terreno.



Figura 2 - Módulos do SIGP

O SIGP inclui também ferramentas avançadas de agricultura de precisão e IoT (*Internet of Things*), permitindo a avaliação espacial do vigor vegetativo, como os índices NDVI e NDRE, através de voos de drones. Esta valência integra ainda sensores e estações meteorológicas para a previsão do risco de doenças e a otimização da rega, facultando o cadastro georreferenciado videira a videira. Adicionalmente, a plataforma incorpora módulos de Inteligência Artificial e proteção de culturas, nomeadamente o *EYESonTRAPS* e o *InOlive*, que exploram algoritmos de visão computacional para a deteção e contagem automática de pragas a partir de imagens reais. O ciclo encerra-se com a gestão da rastreabilidade e transformação agroindustrial, que visa a otimização do fluxo em adegas e lagares, assegurando o controlo total desde a entrada da matéria-prima até ao embalamento e saída comercial do produto final (SIGP, 2026).

1.3 Objetivos e Resultados Esperados

O presente estágio orienta-se por um conjunto de objetivos gerais que visam a aplicação prática dos conhecimentos adquiridos ao longo da formação académica. Em primeiro lugar, procura-se uma integração efetiva em ambiente empresarial, consolidando a autonomia, a capacidade de resolução de problemas e o cumprimento de prazos num contexto real de trabalho. Em segundo lugar, o projeto foca-se na pesquisa do estado da arte e na aplicação técnica, explorando tecnologias emergentes de Inteligência Artificial, nomeadamente os *Large Language Models* (LLM) de execução local e as arquiteturas RAG, para a construção de uma solução plenamente funcional. Como meta final, ambiciona-se um desenvolvimento com impacto prático, no qual o estagiário assume a responsabilidade pela criação de uma componente de *software* com padrões de qualidade para produção (*production-ready*), criando valor direto e mensurável para a plataforma da Geodouro e para os seus utilizadores finais.

Para materializar estes objetivos, perspectiva-se, no final do período de estágio, a entrega de um *chatbot* totalmente funcional e integrado na plataforma SIGP. Este assistente inteligente, garantirá a privacidade absoluta dos dados agrícolas e operacionais da Geodouro. O principal impacto prático esperado é a otimização do suporte ao cliente, reduzindo significativamente o volume de pedidos de assistência técnica mais comuns, ao proporcionar aos utilizadores respostas imediatas e rigorosas com base na documentação interna do sistema.

Do ponto de vista da Engenharia de *Software*, o projeto resultará na entrega do código-fonte com padrões de qualidade para produção (*production-ready*), acompanhado de documentação técnica exaustiva sobre a API desenvolvida e o *pipeline* vetorial. Isto assegurará a futura manutenção e escalabilidade do sistema por parte da equipa da Geodouro. Paralelamente, a nível académico e profissional, espera-se a consolidação de competências avançadas em IA generativa, materializada na elaboração e defesa pública do presente relatório de estágio.

1.4 Plano de Trabalho

O trabalho desenvolvido ao longo das quinze semanas de estágio organizou-se de forma progressiva, com cada etapa a construir a base da seguinte, como se pode ver na Figura 3.

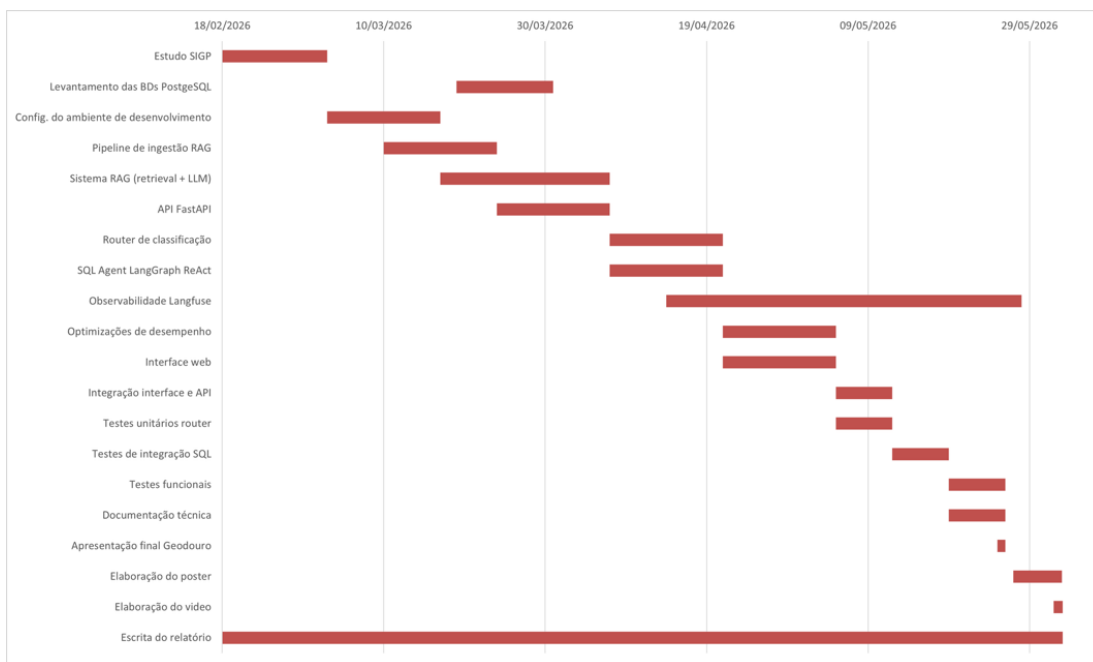


Figura 3 - Gráfico de Gantt

O trabalho desenvolvido organizou-se em cinco fases. Numa fase inicial, procedeu-se ao estudo da plataforma SIGP e ao levantamento e análise das três bases de dados da GeoDouro. Concluída esta etapa, configurou-se o ambiente de desenvolvimento e construiu-se o *pipeline* de ingestão de documentos para vetorização e armazenamento no ChromaDB.

A terceira fase centrou-se no desenvolvimento do sistema, a implementação da API com FastAPI, a integração do modelo Qwen2.5 7B para o sistema RAG, a criação do *router* de classificação por palavras-chave e o desenvolvimento do Agente SQL recorrendo à *framework* LangGraph.

Com o *backend* estável, a quarta fase focou-se na construção da interface web, na integração do sistema de observabilidade Langfuse e na realização de testes exaustivos a três níveis (unitários, de integração e validação funcional). Por fim, o

estágio terminou com a elaboração da documentação técnica e a apresentação final do sistema operacional à equipa da GeoDouro.

1.5 Estrutura do Relatório

O presente relatório encontra-se organizado em seis secções principais, estruturados de forma a refletir a progressão do estágio, desde a fundamentação teórica até aos resultados práticos obtidos.

A secção 1 introduz o enquadramento do projeto, apresentando a entidade de acolhimento, a plataforma SIGP e os objetivos propostos para o desenvolvimento do *chatbot*.

De seguida, a secção 2 expõe o Estado da Arte, focando-se na revisão da arquitetura *Retrieval-Augmented Generation* (RAG), no panorama atual dos *Large Language Models* (LLM) de execução local e na aplicação de Inteligência Artificial na agricultura.

A secção 3 descreve o ecossistema tecnológico empregue, justificando a adoção da *stack* tecnológica selecionada (como Python, FastAPI, ChromaDB e o modelo Qwen2.5 7B) e a sua integração com as tecnologias existentes no SIGP.

O design do sistema, a arquitetura detalhada do *pipeline* RAG e a metodologia de trabalho adotada são apresentados na secção 4.

A secção 5 documenta cronologicamente as atividades desenvolvidas ao longo das quinze semanas de estágio, com detalhe sobre a implementação prática, os desafios técnicos encontrados e as soluções aplicadas em cada fase.

Por fim, a secção 6 apresenta as conclusões do projeto, com avaliação do grau de cumprimento dos objetivos, reflexão sobre o processo de aprendizagem e proposta de trabalhos futuros.

2 Estado da Arte

Este capítulo enquadra teoricamente o projeto através da revisão do estado da arte em quatro eixos fundamentais: arquiteturas *Retrieval-Augmented Generation* (RAG), *Large Language Models* (LLM), bases de dados vetoriais e *embeddings* e Inteligência Artificial aplicada à agricultura de precisão. A análise crítica das soluções existentes permite não só identificar lacunas na literatura atual, mas também fundamentar as opções tecnológicas adotadas e posicionar o contributo desta solução.

2.1 Processamento de Linguagem Natural

O Processamento de Linguagem Natural (PLN) é o ramo da inteligência artificial que estuda a interação entre computadores e a linguagem humana, com o objetivo de capacitar as máquinas a compreender, interpretar e gerar texto de forma significativa (Jurafsky & Martin, 2023). As suas aplicações abrangem desde a tradução automática e a análise de sentimentos até aos sistemas de perguntas e respostas que constituem o fundamento do chatbot desenvolvido neste projeto.

2.1.1 Redes Recorrentes e as suas Limitações

Durante décadas, a abordagem dominante em PLN baseou-se em redes neuronais recorrentes (*Recurrent Neural Networks*, RNN) e nas suas variantes mais sofisticadas, as *Long Short-Term Memory* (LSTM) (Hochreiter & Schmidhuber, 1997) e as *Gated Recurrent Units* (GRU) (Cho et al., 2014). Para serem processados por estas redes, os textos são primeiramente divididos em unidades fundamentais de informação designadas por *tokens*, que podem corresponder a uma palavra inteira, a um fragmento de palavra ou a um único carácter (OpenAI, 2026). Estas arquiteturas

processam o texto de forma estritamente sequencial, *token a token*, mantendo um estado interno que propaga informação ao longo da sequência.

No entanto, as redes recorrentes apresentam duas limitações estruturais que condicionaram o seu progresso. A primeira prende-se com o problema do gradiente desvanescente: em sequências longas, o sinal de aprendizagem propaga-se com dificuldade para os passos iniciais da sequência, fazendo com que o modelo perca a capacidade de relacionar palavras distantes entre si (Cho et al., 2014). Embora arquiteturas posteriores, como as LSTM, atenuem parcialmente este problema, não o eliminam na sua totalidade. A segunda limitação decorre do seu processamento estritamente sequencial. A natureza passo-a-passo das RNN impede a paralelização do processo de treino, fazendo com que o tempo computacional escale linearmente com o comprimento das sequências. Esta restrição arquitetónica torna inviável o treino de modelos de larga escala utilizando o hardware disponível (Vaswani et al., 2017).

2.1.2 A Arquitetura Transformer

Em 2017, Vaswani et al., (2017) propuseram uma arquitetura radicalmente diferente: o *Transformer*. A inovação central desta abordagem é o mecanismo de atenção (*self-attention*), que permite a cada *token* de uma sequência relacionar-se diretamente com todos os outros *tokens* em paralelo, independentemente da distância entre eles. Esta propriedade resolve simultaneamente as duas grandes limitações das redes recorrentes. Por um lado, as dependências de longo alcance passam a ser capturadas de forma direta, eliminando a degradação do sinal de aprendizagem. Por outro lado, a transição para um processamento paralelo de toda a sequência permite escalar o tempo de treino para conjuntos de dados e modelos de dimensão anteriormente considerada inviável (Vaswani et al., 2017). Graças a esta quebra de paradigma, a arquitetura *Transformer* tornou-se o suporte sobre o qual todos os modelos de linguagem modernos são construídos, desde o BERT (Devlin et al., 2019) até às famílias Llama (Meta AI, 2024a) e Qwen (Qwen et al., 2025) avaliadas no âmbito deste projeto.

2.2 Large Language Models

Um Modelo de Linguagem de Grande Escala (*Large Language Model*, LLM) é um sistema de inteligência artificial baseado na arquitetura *Transformer*, treinado em quantidades enormes de texto com o objetivo de modelar a distribuição probabilística da linguagem natural (Vaswani et al., 2017). Em termos práticos, um LLM aprende a prever qual o *token* mais provável a seguir a uma sequência prévia, o que lhe permite gerar texto coerente, responder a perguntas, resumir documentos e seguir instruções em linguagem natural.

Como referido anteriormente, a arquitetura *Transformer*, introduzida por Vaswani et al. (2017), revolucionou o processamento de linguagem natural ao substituir as redes recorrentes por mecanismos de atenção (*self-attention*) que processam toda a sequência de entrada em paralelo. Esta propriedade permite escalar o treino para conjuntos de dados e modelos de dimensão anteriormente inviável, originando os modelos modernos com dezenas ou centenas de mil milhões de parâmetros.

Os LLM modernos são tipicamente treinados em duas fases: um pré-treino não supervisionado que incorpora texto de larga escala (livros, páginas web, código, artigos científicos), seguido de um ajuste fino supervisionado com dados de instruções e, frequentemente, de um processo de alinhamento por reforço com feedback humano (*Reinforcement Learning from Human Feedback*, RLHF) (Ouyang et al., 2022). O resultado são modelos capazes de generalizar para tarefas não vistas durante o treino, incluindo resposta a perguntas em domínios especializados quando complementados com contexto externo como no RAG (*Retrieval-Augmented Generation*) (Lewis et al., 2020) utilizado neste projeto e apresentado na secção 2.2.3.

2.2.1 Parametrização e Quantização Local

A dimensão de um LLM é habitualmente expressa pelo número de parâmetros do modelo, ou seja, os valores numéricos (pesos) aprendidos durante o treino que determinam o comportamento do modelo. Embora os modelos mais recentes e poderosos possuam frequentemente entre 70 e 400 mil milhões de parâmetros (como

é o caso do Llama 3.1 405B (Meta AI, 2024b)), a execução estritamente local em *hardware* de consumo impõe restrições severas de memória.

Por este motivo, o presente projeto foca-se exclusivamente na classe de modelos otimizados para execução local, cujo limite prático para inferência numa placa gráfica de 8GB de VRAM (*Video Random Access Memory*, a memória de vídeo dedicada da unidade de processamento gráfico responsável por armazenar os dados durante a execução) situa-se na casa dos 8 mil milhões de parâmetros. Estes valores são habitualmente representados pela letra B (do inglês *billion*). Na presente avaliação, foram selecionados modelos inseridos em três patamares dimensionais: os de 3B, como o StableLM-Zephyr 3B (Stability AI, 2023), os de 7B, exemplificados pelo Mistral 7B (Jiang et al., 2023) e pelo Qwen2.5 7B (Qwen et al., 2025), e finalmente, os que operam no limite prático imposto pelo *hardware*, com 8B, onde se inserem o Llama 3.1 8B (Meta AI, 2024a) e o Qwen3-8B (Qwen Team, 2025).

A gestão destes parâmetros é crítica, uma vez que cada um ocupa espaço em memória. Um modelo em precisão total requer 4 bytes por parâmetro, pelo que uma rede de 7B ocuparia aproximadamente 28GB. Para contornar esta barreira e tornar estes modelos executáveis no *hardware* disponível, recorre-se à quantização, uma técnica que reduz a precisão numérica dos pesos de 32 bits para 4 ou 8 bits, com perda mínima de qualidade (Frantar et al., 2023). Na quantização Q4_K_M utilizada neste projeto, cada parâmetro é representado com cerca de 4 bits em linha com os esquemas de quantização 4-bit weight-only descritos por Frantar et al., (2023), o que permite comprimir um modelo de 7B parâmetros para cerca de 4 GB de pesos, valor que na prática se traduz em aproximadamente 4–5 GB de VRAM ocupada durante a execução.

Contudo, a relação entre o número de parâmetros e a qualidade da geração não é linear nem universal. Investigação recente demonstrou que modelos mais pequenos, treinados em dados de maior qualidade e durante mais tempo, superam frequentemente alternativas maiores treinadas de forma menos eficiente, um fenómeno explorado pela família Mistral (Jiang et al., 2023) e confirmado pelos resultados empíricos deste projeto. Acresce ainda que a contagem de parâmetros

pode não refletir a carga computacional direta, como sucede nos modelos de arquitetura *Mixture-of-Experts* (MoE) (Shazeer et al., 2017). O DeepSeek-V2-Lite (DeepSeek-AI, 2024), por exemplo, possui 15.7B parâmetros totais, mas apenas 2.4B são ativados por inferência, pois cada *token* é encaminhado unicamente para um subconjunto de especialistas, uma distinção fulcral na estimativa da VRAM estritamente necessária.

2.2.2 Limitações dos LLM e a Necessidade de Conhecimento Externo

Apesar das capacidades notáveis dos LLM, a sua utilização em sistemas de suporte a domínios específicos, como o chatbot do SIGP, expõe três limitações fundamentais que inviabilizam a sua aplicação direta sem uma arquitetura complementar. A primeira prende-se com o seu conhecimento estritamente estático, uma vez que os modelos são treinados com dados até uma data de corte específica (*knowledge cutoff*). Consequentemente, qualquer informação posterior a essa data, incluindo atualizações de *software*, novos procedimentos ou alterações funcionais, permanece totalmente desconhecida para o modelo (Gao et al., 2024). A esta restrição temporal junta-se a ausência de conhecimento de domínio privado. A documentação interna de um sistema corporativo como o SIGP nunca integrou os conjuntos de dados de treino de nenhum modelo público, deixando o LLM sem qualquer contexto sobre os menus, procedimentos operacionais ou terminologia específica da aplicação. Por último, surge o desafio crítico das alucinações. Quando confrontado com uma pergunta para a qual não possui informação residente suficiente, um LLM tende a gerar respostas estruturalmente plausíveis, mas factualmente incorretas (Ji et al., 2023). Num ecossistema de suporte agrícola, este comportamento é particularmente grave, pois respostas erradas sobre procedimentos delicados, como a gestão de pragas, podem induzir o utilizador a tomar decisões com consequências reais. É precisamente para mitigar este triplo obstáculo que surge o paradigma RAG (*Retrieval-Augmented Generation*), detalhado na secção seguinte.

2.2.3 Janela de Contexto e RAG

A janela de contexto (*context window*) de um LLM define o número máximo de *tokens* que o modelo consegue processar numa única inferência, englobando tanto no texto de entrada (*prompt*) como na resposta gerada. De acordo com a OpenAI, um *token* corresponde tipicamente a 0,75 palavras em inglês (OpenAI, 2026). Contudo, devido à mecânica dos algoritmos de tokenização (*Byte-Pair Encoding*), línguas como o português sofrem uma ligeira penalização, exigindo frequentemente entre 1,5 a 2 *tokens* por palavra (Ahia et al., 2023). Neste cenário conservador, uma janela de 4096 *tokens* equivale a cerca de 2000 a 2500 palavras em português, enquanto uma janela de 128k *tokens* acomoda aproximadamente 96000 palavras (OpenAI, 2026).

Esta limitação dimensional assume particular relevância no paradigma RAG (*Retrieval-Augmented Generation*) (Lewis et al., 2020), que constitui a arquitetura base do *chatbot* desenvolvido neste projeto. Em geral, o RAG é uma técnica que resolve a falta de conhecimento específico dos LLMs ao atuar como uma ponte para uma base de documentação externa. Uma vez que o modelo não possui conhecimento interno sobre a plataforma corporativa, o sistema tem de pesquisar e recuperar os documentos mais relevantes a cada interação do utilizador. Para este efeito, o sistema recorre a uma base de dados vetorial (uma tecnologia otimizada para pesquisa semântica cujo funcionamento será detalhado na secção 2.3), extraíndo os excertos necessários (Gao et al., 2024) e inserindo-os diretamente no *prompt* enviado ao LLM. No âmbito deste projeto, este *prompt* é composto por três elementos distintos que competem diretamente pelo espaço finito da janela de contexto. Em primeiro lugar, integra-se o *prompt* de sistema, que define as instruções de comportamento do *chatbot* (como a língua, o tom e as restrições operacionais), ocupando nesta solução tipicamente entre 100 a 300 *tokens*. A este junta-se o bloco dos *chunks* recuperados, correspondente aos excertos documentais relevantes para a pergunta. Ao adotar um parâmetro de $k=3$ *chunks* com cerca de 400 *tokens* cada, este segmento consome aproximadamente 1200 *tokens*. Por fim, incorpora-se a pergunta atual do utilizador e, opcionalmente, o histórico de mensagens anteriores da conversa.

Nesta complexa gestão de memória, uma janela de contexto de 4096 *tokens* (como a do StableLM-Zephyr 3B) revela-se insuficiente. Subtraindo a carga do *prompt* de sistema, dos *chunks* e da pergunta, o espaço restante para a geração da resposta é inferior a 1000 *tokens*, o que limita severamente a capacidade explicativa do modelo e inviabiliza a manutenção de um histórico conversacional útil. Em contrapartida, modelos equipados com janelas de 32k ou 128k *tokens* eliminam esta restrição arquitetônica, permitindo recuperar mais *chunks*, acomodar um contexto conversacional alargado e gerar respostas manifestamente mais completas (Gao et al., 2024).

Importa ressaltar, contudo, que a relação entre a amplitude da janela de contexto, o número de *chunks* recuperados e a qualidade final da resposta RAG não é estritamente proporcional. Conforme demonstrado por Gao et al. (2024), o aumento do parâmetro *k* melhora a qualidade da resposta apenas até um determinado ponto de saturação. Ultrapassado esse limiar, a introdução de informação redundante ou irrelevante começa a degradar o foco e o desempenho do modelo, um fenômeno documentado na literatura como *lost in the middle* (Liu et al., 2023). Esta evidência empírica fundamenta a escolha conservadora de *k*=3 adotada nos testes deste projeto, um valor que procura o equilíbrio ótimo entre a cobertura de informação necessária e a mitigação de ruído no contexto fornecido ao modelo.

2.3 Bases de Dados Vetoriais e Embeddings

O processamento de linguagem natural por modelos computacionais exige a conversão prévia de texto não estruturado em representações matemáticas. Os *embeddings* modernos cumprem esta função vital, mapeando documentos num espaço vetorial multidimensional onde a proximidade geométrica reflete a similaridade semântica (Nie et al., 2025). Com a explosão arquitetônica do paradigma RAG em 2024, os sistemas de gestão de bases de dados relacionais tradicionais revelaram-se estruturalmente ineficazes na pesquisa e recuperação rápida destas matrizes de alta dimensão. Para combater este desafio, emergiram as bases de dados vetoriais (*Vector Databases*), otimizadas especificamente para o armazenamento e

consulta de *embeddings* à escala industrial (Ma et al., 2025). Ao invés de procurarem correspondências exatas por palavras-chave, estas bases recorrem a algoritmos avançados de pesquisa aproximada de vizinhos mais próximos (*Approximate Nearest Neighbor*, ANN), fundamentais para contornar os estrangulamentos de latência nos LLM modernos (Ma et al., 2025). Num sistema RAG, é esta tecnologia que permite comparar instantaneamente o vetor da pergunta com milhares de vetores de documentos, assumindo o papel de motor de memória de longo prazo do assistente virtual.

2.4 Agentes Conversacionais (Chatbots)

A evolução das interfaces de interação humano-computador sofreu uma mudança de paradigma drástica com a transição dos *chatbots* baseados em regras para os agentes conversacionais suportados por Inteligência Artificial generativa (Gupta et al., 2024). Historicamente, os sistemas de suporte automatizado dependiam de árvores de decisão rígidas e guiões pré-programados. Estes sistemas tradicionais revelavam-se frequentemente frustrantes para os utilizadores finais, uma vez que falhavam na compreensão de variações linguísticas, erros ortográficos ou intenções complexas que saíssem fora do fluxo estritamente mapeado pelos programadores (Gupta et al., 2024).

Com o advento dos LLM, os *chatbots* modernos transformaram-se em agentes dinâmicos com elevada capacidade de raciocínio natural (Gupta et al., 2024). Em vez de selecionarem uma resposta de uma base de dados pré-concebida, estes agentes geram texto de forma probabilística e contextualizada em tempo real (Gupta et al., 2024). No contexto de plataformas de gestão complexas, como o SIGP da Geodouro, a adoção de uma interface conversacional fluida reduz substancialmente a curva de aprendizagem. Permite que agricultores e técnicos interajam com o sistema utilizando linguagem natural, formulando perguntas abertas sobre regulamentações ou funcionalidades de *software*, em vez de navegarem exaustivamente por menus ou manuais de utilizador extensos.

2.5 Aplicações de IA na Agricultura

O setor agrícola encontra-se num processo de profunda transformação digital, frequentemente designado por Agricultura 4.0. Nesta transição para uma agricultura de precisão, a adoção de sistemas de apoio à decisão baseados em Inteligência Artificial tem-se revelado fundamental para otimizar a gestão de recursos e mitigar os impactos de variáveis imprevisíveis (Paul & Machavaram, 2025). Atualmente, a IA já possui uma forte implementação em tarefas de visão computacional, como é o caso dos módulos EYESonTRAPS e InOlive desenvolvidos pela Geodouro para a deteção de pragas. Contudo, a aplicação de Processamento de Linguagem Natural avançado no apoio à decisão agronómica e na usabilidade de *software* agrícola encontra-se ainda numa fase inicial (Mittal et al., 2025).

A literatura mais recente demonstra que a simples aplicação de LLM generalistas (como o ChatGPT) não é suficiente para o contexto agrícola especializado (Didwania et al., 2024). Estes modelos carecem frequentemente de conhecimentos técnicos hiperespecializados (Sapkota et al., 2025) e da capacidade de adaptação a regulamentações locais em constante mutação, como as diretivas da Política Agrícola Comum (PAC) (GPP, 2026). Para dar resposta a este desafio técnico, têm surgido abordagens experimentais baseadas em RAG desenhadas especificamente para o domínio agroindustrial (Fanuel et al., 2025). A implementação de mecanismos de recuperação de informação sensíveis à região (*region-aware retrieval*) e a injeção de documentação corporativa privada permitem gerar recomendações de alta-fidelidade. É neste panorama de inovação que o presente estágio se insere, procurando transpor estas abordagens teóricas de IA generativa para um produto comercial escalável e totalmente executado de forma local.

2.6 Lacunas e Contribuição

A revisão bibliográfica efetuada evidencia o enorme potencial da Inteligência Artificial, nomeadamente das arquiteturas RAG, na modernização do setor agrícola. No entanto, a análise crítica do panorama atual permite identificar lacunas

significativas na transição destas tecnologias para o tecido empresarial corporativo. A primeira lacuna prende-se com a dependência tecnológica e a privacidade da informação. A maioria das soluções de agentes conversacionais documentadas recorre a Modelos de Linguagem alojados na *cloud* (como as API comerciais da OpenAI). No domínio agrícola, onde os dados de produção e a gestão do território são estritamente confidenciais, esta dependência externa representa um risco inaceitável de exposição de dados e de perda de soberania. A esta vulnerabilidade acresce o desafio do custo e da escalabilidade. A adoção de soluções comerciais baseadas num modelo de faturação por *tokens* consumidos introduz encargos recorrentes e imprevisíveis, o que dificulta severamente a expansão e democratização do *software* para múltiplos produtores. Por fim, denota-se uma clara falta de contexto local. Os modelos generalistas tendem a sofrer de alucinações quando confrontados com terminologia agronómica específica ou com a complexa quantidade de regulamentações locais, exigindo um contexto rigoroso e sistematicamente adaptado à realidade do produtor.

O presente projeto de estágio posiciona-se exatamente na interseção destas lacunas. A principal contribuição consiste na concetualização e desenvolvimento de um assistente inteligente agrícola, totalmente integrado no Sistema Integrado de Gestão de Propriedades (SIGP), que ultrapassa as limitações identificadas através de uma arquitetura estritamente ancorada em tecnologias *open-source* e de execução local.

Ao implementar um sistema RAG suportado por um modelo de linguagem de excelência técnica e uma base de dados vetorial integrada, o projeto assegura três eixos de valor. Em primeiro lugar, garante a privacidade absoluta, uma vez que os dados sensíveis dos utilizadores do SIGP nunca abandonam a infraestrutura corporativa. Em segundo lugar, promove a isenção de custos recorrentes, pois a adoção de uma *stack* totalmente *open-source* elimina os encargos de licenciamento e a faturação por *tokens* consumidos em API externas. Por fim, assegura a geração de respostas de alta fidelidade: a restrição do conhecimento do modelo exclusivamente à documentação validada do SIGP mitiga o risco de alucinações, garantindo respostas

precisas, técnicas e contextualmente adequadas à realidade regulamentar e agronómica portuguesa.

2.7 Resumo

Este capítulo fundamenta as escolhas tecnológicas do projeto, destacando a utilização de Modelos de Linguagem otimizados para execução local. Como as inteligências artificiais genéricas desconhecem a realidade agrícola, as regras regulamentares e o próprio software da empresa, justifica-se a adoção da arquitetura RAG e de bases de dados vetoriais, que obrigam o sistema a ler os manuais oficiais antes de responder ao utilizador. A grande contribuição deste estágio materializa-se, assim, na criação de um *chatbot* para o SIGP que é completamente privado e seguro, isento de custos mensais por utilizar apenas tecnologias open-source, e capaz de fornecer aos agricultores um suporte técnico fiável sem que os dados saiam da empresa.

3 Tecnologias e Ferramentas Utilizadas

Este capítulo detalha a arquitetura tecnológica selecionada para a concepção e desenvolvimento do *chatbot* agrícola. As escolhas aqui apresentadas resultam da fundamentação teórica explorada no estado da arte, em conjunto aos requisitos do projeto: a garantia de confidencialidade dos dados operacionais através de processamento local, a adoção preferencial de soluções de código aberto (*open-source*) e a necessidade de uma integração fluída com o ecossistema tecnológico preexistente na entidade de acolhimento. A descrição encontra-se segmentada nas camadas lógicas do sistema, desde o *backend* até às ferramentas de suporte ao desenvolvimento.

3.1 Backend

O *backend* constitui o núcleo do sistema, sendo responsável pela orquestração do fluxo de dados entre a base de conhecimento, o modelo de linguagem e a interface do utilizador. As tecnologias selecionadas para o *backend* priorizam o desempenho de processamento, a escalabilidade e a compatibilidade nativa com o ecossistema moderno de Inteligência Artificial.

3.1.1 Python e FastAPI

A adoção da linguagem Python para o desenvolvimento do módulo de Inteligência Artificial justifica-se pelo seu estatuto consolidado como o padrão da indústria nos domínios de *Machine Learning* e Processamento de Linguagem Natural (Raschka et

al., 2020). O ecossistema Python providencia as bibliotecas e estruturas de desenvolvimento essenciais para a concretização de agentes conversacionais, evitando a necessidade de construir ferramentas complexas de raiz (Wolf et al., 2020).

Para a disponibilização deste núcleo sob a forma de uma Interface de Programação de Aplicações (API), selecionou-se a *framework* FastAPI (FastAPI, 2026). A escolha desta tecnologia, ao contrário de alternativas como o Flask (menos otimizado para concorrência) ou o Django (demasiado monolítico para o âmbito do projeto) (Comparison of FastAPI with Django and Flask, 2025), fundamenta-se em três pilares técnicos.

O primeiro diz respeito ao processamento assíncrono nativo. Os sistemas baseados em RAG são limitados por operações de entrada e saída, como a consulta a bases de dados vetoriais e a espera pela inferência do LLM. O suporte nativo do FastAPI para rotinas assíncronas, assente no módulo *asyncio* do Python (The Python Standard Library, 2026), permite processar múltiplos pedidos em simultâneo sem bloquear os recursos do servidor, necessário para o desempenho do *chatbot*.

O segundo pilar é a validação de dados. Através da integração com a biblioteca Pydantic (Pydantic Validation, 2026) o FastAPI valida automaticamente a estrutura dos pedidos e respostas da API, o que garante a conformidade dos dados trocados e reduz os erros em tempo de execução.

O terceiro pilar é a documentação automática. A geração nativa de documentação interativa segundo a especificação OpenAPI (OpenAPI Specification, 2021) acelera de forma considerável as fases de teste e simplifica a integração posterior com o frontend.

3.1.2 LangChain e LangGraph

LangChain (LangChain, 2026) é uma *framework* open-source para o desenvolvimento de aplicações baseadas em modelos de linguagem, fornecendo abstrações para encadear operações como recuperação de documentos, geração de texto e interação

com ferramentas externas. LangGraph (LangGraph, 2026) é uma extensão do LangChain orientada à construção de agentes com grafos de estado, permitindo fluxos de raciocínio multi-passo.

A escolha de LangChain justifica-se pela maturidade do ecossistema, pela integração nativa com Ollama e ChromaDB (Chroma, 2026), e pela disponibilidade de componentes pré-construídos para *pipelines* RAG e agentes ReAct, um conceito introduzido por Yao et al. (2022).

Os principais componentes utilizados pertencem ao ecossistema LangChain e LangGraph (LangChain, 2026; LangGraph, 2026). O “ConversationalRetrievalChain” implementa o pipeline RAG com memória conversacional e assegura a recuperação de documentos com contexto de sessão. O “create_react_agent”, disponibilizado pelo LangGraph, é responsável pela criação do agente SQL segundo o padrão ReAct com *tool calling*. O “PromptTemplate” possibilita a definição de *prompts* estruturados com variáveis dinâmicas, enquanto o “ConversationBufferWindowMemory” gere a memória temporária da sessão e retém apenas as últimas mensagens da conversa.

3.1.3 ChromaDB

ChromaDB é uma base de dados vetorial *open-source*, concebida para armazenar e pesquisar vetores de *embeddings* de forma eficiente (Chroma, 2026). Suporta persistência em disco, pesquisa por similaridade e filtragem por metadados.

As bases de dados vetoriais são fundamentais em sistemas RAG, pois permitem encontrar os segmentos de documentação mais relevantes para uma dada pergunta, através da comparação de representações vetoriais do texto (Gao et al., 2024).

No projeto, o ChromaDB foi utilizado para armazenar os *embeddings* da documentação relativamente ao SIGP, indexar os nomes e colunas das tabelas das bases de dados e agrupar as mesmas em domínios de forma a melhorar a precisão de pesquisa de tabelas.

3.1.4 Ollama

O Ollama é uma plataforma de código aberto para execução local de modelos de linguagem de grande escala (LLM), que permite correr modelos como Llama, Mistral e Qwen em hardware convencional sem necessidade de infraestrutura cloud (Huang & Wang, 2025). A plataforma gere automaticamente a transferência de modelos e a alocação de recursos de CPU e GPU.

A escolha do Ollama justifica-se por três fatores (Huang & Wang, 2025). O primeiro é a privacidade de dados, uma vez que os dados do SIGP nunca saem do servidor local. O segundo é o custo zero de operação, dado que não existem subscrições de APIs comerciais associadas. O terceiro é a independência de conectividade, pois o sistema funciona totalmente offline após a transferência inicial dos modelos.

3.1.5 Qwen2.5

O Qwen2.5 é uma família de modelos de linguagem desenvolvida pelo Alibaba Cloud, disponível em várias dimensões. Os modelos destacam-se pelo suporte a múltiplas línguas (incluindo o português europeu), pela capacidade de *tool calling* (invocação de ferramentas com saída estruturada) e pela eficiência em tarefas de geração de código e SQL (Qwen et al., 2025).

Para selecionar o modelo a utilizar no projeto, foram avaliados oito candidatos no hardware disponível, com janela de contexto de 1024 *tokens*: Qwen2.5 7B Instruct, Qwen3 8B, Qwen2.5-VL 7B Instruct, Llama 3.1 8B Instruct, Mistral 7B, Mistral NeMo

12B, DeepSeek-V2-Lite e StableLM-Zephyr 3B. Os resultados do *benchmark* encontram-se na Figura 4.

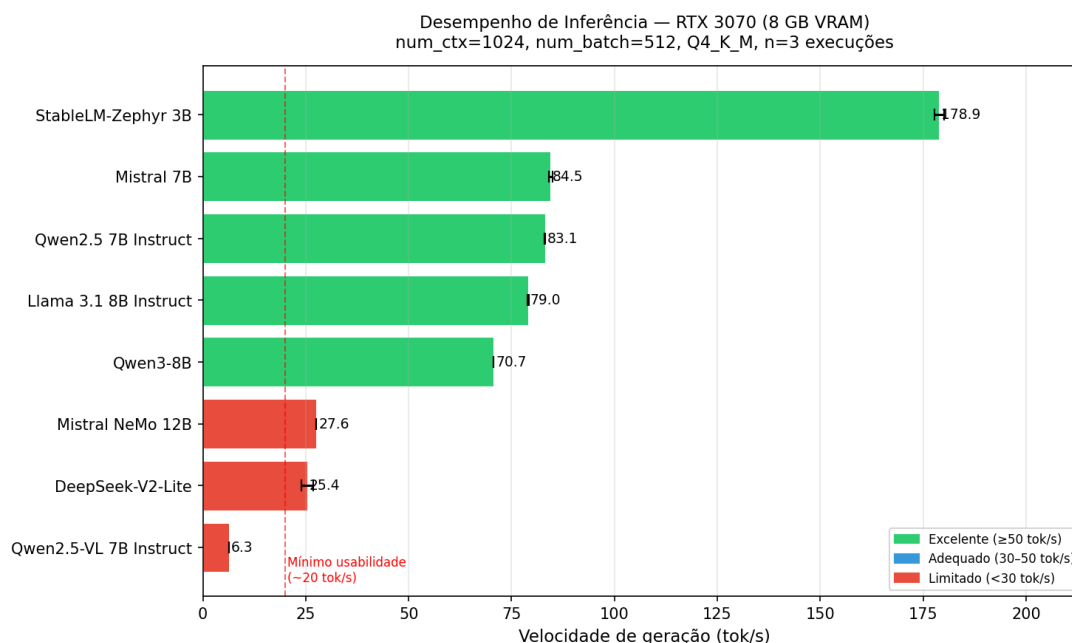


Figura 4 - Desempenho das várias LLM

O Qwen2.5 7B Instruct foi o modelo selecionado por apresentar o melhor equilíbrio entre qualidade das respostas em português, capacidade de *tool calling* e velocidade de inferência. Este modelo também foi selecionado pela GeoDouro. O StableLM-Zephyr 3B foi o mais rápido (178,9 tok/s), mas insuficiente para tarefas de raciocínio multi-passo, tal como o agente SQL. O Mistral 7B obteve velocidade ligeiramente superior (84,5 tok/s contra 83,1 tok/s do Qwen2.5), porém com pior desempenho na língua portuguesa, realizando alucinações no seguimento de instruções complexas.

No projeto, o Qwen2.5 7B Instruct é utilizado nos três componentes que invocam o modelo de linguagem: o pipeline RAG, o agente SQL e a condensação de perguntas com histórico conversacional. A utilização de uma única variante em todos os componentes elimina recarregamentos do servidor de modelos Ollama entre chamadas consecutivas, com impacto direto na latência do sistema como poderá ser visto na secção 5.4.3.

3.1.6 HuggingFace Embeddings

Os modelos de *embeddings* transformam texto em representações vetoriais de dimensão fixa, permitindo a medição de similaridade semântica entre partes de texto (Nie et al., 2025). O projeto utiliza o modelo *intfloat/multilingual-e5-small*, um modelo compacto e eficiente que gera vetores de 384 dimensões (Wang et al., 2024). Este modelo foi escolhido pelo equilíbrio entre qualidade de representação semântica, velocidade de processamento, compatibilidade com várias línguas e consumo de memória, fatores críticos num sistema que necessita de indexar centenas de tabelas e documentos em hardware local.

3.1.7 Psycopg2

Psycopg2 é o adaptador PostgreSQL mais utilizado em Python, fornecendo uma implementação completa com suporte a transações e tipos de dados avançados (Psycopg, 2026).

O *psycopg2* é utilizado pelo agente SQL para estabelecer ligações às bases de dados PostgreSQL e executar *queries* SELECT em modo *read-only*. As ligações são configuradas com *timeout* e modo somente leitura como medidas de segurança.

3.2 Frontend

O desenvolvimento da interface com o utilizador procurou responder a um requisito do projeto, a criação de um ambiente de interação focado, limpo e independente. Em vez de se embutir o assistente nas janelas pré-existentes do SIGP, optou-se pela conceção de uma página web dedicada. Esta arquitetura apresenta vantagens significativas, nomeadamente a separação clara de responsabilidades. Ao operar de forma autónoma, a interface do chatbot não consome recursos de processamento da plataforma principal de gestão agrícola, oferecendo ao utilizador um espaço de ecrã inteiro otimizado para a leitura de respostas extensas, tabelas de dados e documentação técnica gerada pelo modelo.

A implementação do comportamento dinâmico desta página independente baseou-se inteiramente em JavaScript puro, dispensando a adoção de *frameworks* pesadas como o React ou o Angular. Esta decisão técnica foi motivada pela necessidade de garantir uma aplicação extremamente leve, de carregamento instantâneo e de fácil manutenção futura por parte da equipa da empresa. O JavaScript atua como o motor lógico da página, sendo responsável por manipular o DOM (*Document Object Model*) em tempo real, capturar as interações do utilizador, efetuar os pedidos assíncronos à infraestrutura *backend* e mostrar fluidamente as respostas textuais processadas pelo modelo de linguagem.

3.3 Ferramentas de Desenvolvimento

Para suportar todo o ciclo de vida do desenvolvimento de software, desde a fase inicial até à otimização através de testes e integração final, selecionou-se um conjunto de ferramentas consolidadas na indústria tecnológica. A adoção destas plataformas permitiu assegurar a organização estrutural do projeto, a estabilidade do código e a validação contínua dos vários componentes do sistema.

3.3.1 Git/GitHub

O Git (*Git*, 2026) foi adotado como o sistema de controlo de versões primário do projeto, viabilizando o registo cronológico e rigoroso de todas as alterações efetuadas no código. O GitHub (*GitHub*, 2026) operou como a plataforma de alojamento remoto do repositório, assegurando, não apenas a salvaguarda de todo o trabalho desenvolvido, mas também a documentação iterativa da evolução da arquitetura, facilitando a partilha e auditoria de código por parte da equipa técnica da empresa.

3.3.2 VSCode

O Visual Studio Code (*Visual Studio Code*, 2026) foi o ambiente de desenvolvimento integrado eleito para a realização do estágio. A sua escolha justificou-se pela sua leveza de execução e pelo ecossistema modular de extensões. Esta flexibilidade

permitiu centralizar numa única interface a edição multilinguagem, abrangendo desde o Python, JavaScript e HTML/CSS para a interface, e ficheiros .env de configuração, bem como a integração de terminais locais para a gestão direta do servidor FastAPI e do motor Ollama.

3.3.3 Postman

Atendendo à natureza distribuída do sistema (baseado na comunicação entre a interface visual e uma API), o Postman (*Postman*, 2026) assumiu um papel crítico durante as fases intermédias de desenvolvimento. Esta ferramenta foi utilizada para testar os *endpoints* gerados pelo FastAPI antes da sua integração no *frontend*. Permitiu simular cenários reais de interação, injetando *payloads* em formato JSON, e forneceu métricas detalhadas sobre o formato das respostas, a gestão de erros como, códigos HTTP, e os tempos de latência gerados pelas respostas do LLM.

3.3.4 LangFuse

O Langfuse é uma plataforma de observabilidade e avaliação para aplicações baseadas em modelos de linguagem, de código aberto, disponível em modo *cloud* ou em modo *self hosted* (*Langfuse*, 2026). Neste contexto a observabilidade refere-se à capacidade de inspecionar o funcionamento interno do sistema a partir de dados registados externamente, sem alterar a lógica da aplicação. A plataforma regista cada execução do sistema sob a forma de *traces*, registos detalhados que incluem todos os passos intermédios como chamadas ao modelo, ferramentas invocadas, *inputs*, *outputs*, latências por componente e *tokens* consumidos. Estes dados ficam disponíveis numa interface web navegável que permite inspecionar o comportamento do sistema ao nível de cada pedido individual.

A integração com *frameworks* como o LangChain e o LangGraph é feita através de um *callback handler* que instrumenta automaticamente as *chains* e os agentes, sem alterações à lógica. O Langfuse oferece ainda funcionalidades de avaliação de qualidade (*scores*), gestão de *prompts* com diferentes versões e criação de conjuntos de dados para testes de regressão.

No projeto, o Langfuse foi instalado em modo *self hosted* devido ao requisito de privacidade estabelecido pela empresa, que impede o envio de dados para serviços externos. É utilizado para monitorizar todos os pedidos ao *endpoint* “/chat”, com *traces* que cobrem o *pipeline* RAG, o agente SQL ReAct e o componente de condensação de perguntas. A análise dos traces foi determinante no diagnóstico de problemas de desempenho e de comportamento do agente SQL durante o desenvolvimento descrito na secção 5.4.2.

3.4 Resumo

A Tabela 1 - Tecnologias seleccionadas para cada camada sintetiza as tecnologias seleccionadas para cada camada do sistema, com a respetiva justificação de adoção.

Tabela 1 - Tecnologias seleccionadas para cada camada

Camada	Tecnologia	Papel no sistema
Backend	Python + FastAPI	Núcleo do sistema e exposição da API com suporte assíncrono e streaming
Orquestração	LangChain + LangGraph	Pipeline RAG e agente SQL com padrão ReAct
Vectorstore	ChromaDB	Armazenamento e pesquisa de embeddings de documentação e tabelas
Modelos locais	Ollama + Qwen2.5 7B	Execução local do LLM sem envio de dados para serviços externos
Embeddings	multilingual-e5-small	Representação semântica de texto em português

Base de dados	Psycopg2 + PostgreSQL	Ligação às três bases de dados operacionais em modo leitura
Frontend	HTML + CSS + JavaScript	Interface web com streaming de respostas
Controlo de versões	Git + GitHub	Registo de alterações e partilha de código
Desenvolvimento	VSCode + Postman	Edição e teste dos endpoints da API
Observabilidade	Langfuse (self-hosted)	Registo de traces, diagnóstico e feedback de utilizador

A seleção privilegiou soluções *open-source* executáveis localmente, em resposta ao requisito de privacidade da empresa. Nenhum dado operacional sai do servidor durante o funcionamento normal do sistema.

4 Metodologia e Conceção

Este capítulo descreve a metodologia de trabalho adotada ao longo do estágio e a conceção da solução desenvolvida. Numa primeira parte, apresenta-se a abordagem iterativa utilizada para organizar o desenvolvimento e o processo de acompanhamento com o orientador. De seguida, definem-se os requisitos funcionais e não funcionais do sistema, os *user stories* que os complementam e o diagrama de casos de uso. A parte final descreve a arquitetura da solução, com detalhe sobre os componentes principais como o router de classificação, o pipeline RAG, o SQL Agent e o sistema de observabilidade.

4.1 Metodologia de Trabalho

O desenvolvimento do chatbot seguiu uma metodologia ágil simplificada, adaptada ao contexto de estágio curricular. Dado que o projeto foi desenvolvido por um único estagiário com acompanhamento do orientador na empresa, não se justificava adotar na íntegra um framework como Scrum ou Kanban, que pressupõem equipas multidisciplinares e papéis formais (*Scrum Guides*, 2026). Optou-se, em alternativa, por uma abordagem iterativa e incremental inspirada nos princípios do Manifesto Ágil (*Manifesto for Agile Software Development*, 2026).

4.1.1 Metodologia Ágil

A metodologia ágil, formalizada pelo Manifesto Ágil, propõe um conjunto de valores que privilegiam a entrega contínua de software funcional, a capacidade de resposta à mudança e a colaboração próxima entre as partes envolvidas. As abordagens ágeis são particularmente adequadas a projetos onde os requisitos podem evoluir ao longo do desenvolvimento, o que se aplica ao presente caso. Como se trata de um projeto

exploratório, muitas decisões técnicas só puderam ser tomadas após experimentação prática.

A adaptação ao contexto de estágio consistiu em organizar o trabalho em ciclos curtos, cada um com um objetivo concreto. No início de cada ciclo definia-se o que implementar e no final, o resultado era testado e discutido com o orientador. Como suporte a este processo de acompanhamento, elaboraram-se relatórios técnicos de progresso destinados exclusivamente à equipa interna da GeoDouro. Estes documentos detalham as descobertas de cada ciclo e métricas de desempenho e encontram-se nos anexos deste relatório.

A priorização seguiu o critério de impacto na qualidade das respostas e na experiência do utilizador. O plano de trabalho foi ajustado várias vezes com base em descobertas técnicas, como por exemplo, a integração de observabilidade revelou problemas de desempenho não previstos inicialmente, o que levou a reordenar prioridades e corrigir o tempo de resposta de aproximadamente 60 segundos para cerca de 3 segundos antes de avançar para novas funcionalidades.

4.1.2 Acompanhamento e Validação

O acompanhamento foi realizado através de reuniões semanais com o orientador na empresa. Cada reunião incluía a apresentação do trabalho realizado, frequentemente com demonstrações do chatbot, a discussão de dificuldades técnicas e a definição de prioridades para a semana seguinte. Este formato permitiu manter o alinhamento entre o trabalho técnico e as necessidades da empresa.

4.2 Requisitos

O levantamento de requisitos foi realizado de forma progressiva ao longo do estágio, com base nas reuniões semanais com o orientador e na análise das necessidades dos utilizadores do SIGP. Os requisitos foram refinados iterativamente à medida que o sistema evoluía e se tornava possível validar funcionalidades com perguntas reais.

4.2.1 Requisitos Funcionais

Os requisitos funcionais descrevem as capacidades que o sistema deve oferecer aos seus utilizadores e foram identificados com base nas necessidades da empresa, tendo sido validados através de demonstrações práticas ao longo do estágio. A Tabela 2 resume os requisitos funcionais do *chatbot*.

Tabela 2 - Requisitos Funcionais

Nº	Requisito
RF1	O sistema deve permitir ao utilizador fazer perguntas em linguagem natural, em português de Portugal, através de uma interface web.
RF 2	O sistema deve classificar automaticamente cada pergunta e encaminhá-la para o pipeline adequado (documentação, base de dados, ou ambos).
RF 3	O sistema deve responder a perguntas sobre as funcionalidades do SIGP com base na documentação existente, utilizando pesquisa semântica (RAG).
RF 4	O sistema deve traduzir perguntas em linguagem natural para <i>queries</i> SQL e consultar as bases de dados PostgreSQL do SIGP.
RF 5	O sistema deve manter o contexto da conversa dentro de uma sessão, permitindo perguntas de seguimento (por exemplo, "e qual a mínima?" após uma pergunta sobre áreas).
RF 6	Quando a pergunta requer simultaneamente informação da documentação e da base de dados, o sistema deve combinar ambas as fontes numa resposta coerente.
RF 7	O sistema deve declarar explicitamente quando não dispõe de informação suficiente para responder, em vez de gerar informação incorreta.

RF 8	O sistema deve apresentar resultados tabulares em formato de tabela e listagens em formato de lista, adaptando a formatação ao tipo de resposta.
RF 9	O sistema deve responder a saudações e mensagens de conversa informal de forma adequada, sem invocar os <i>pipelines</i> de RAG ou SQL.

4.2.2 Requisitos Não Funcionais

Os requisitos não funcionais definem as restrições e qualidades transversais do sistema. O requisito de privacidade (RNF 1) foi estabelecido como prioritário pela empresa desde o início do estágio e condicionou várias decisões arquiteturais, nomeadamente a escolha de modelos locais e a instalação local da plataforma de observabilidade. A Tabela 3 resume os requisitos não funcionais identificados.

Tabela 3 - Requisitos Não Funcionais

Nº	Requisito	Descrição
RNF 1	Privacidade	Todos os dados devem ser processados localmente, sem envio para serviços externos. Os modelos de linguagem e a plataforma de observabilidade devem ser executados na infraestrutura da empresa.
RNF 2	Desempenho	O tempo de resposta para perguntas de documentação (RAG) deve ser inferior a 5 segundos após o carregamento inicial do modelo.
RNF 3	Desempenho	O tempo de resposta para consultas à base de dados (SQL) deve ser inferior a 15 segundos, incluindo a geração e execução da <i>query</i> .
RNF 4	Fiabilidade	O sistema deve evitar alucinações, respondendo apenas com base em dados verificáveis

		(documentação indexada ou resultados de <i>queries</i> SQL).
RNF 5	Usabilidade	As respostas devem ser formuladas em linguagem simples e natural, sem recurso a termos técnicos como "tabelas", " <i>queries</i> " ou "base de dados".
RNF 6	Portabilidade	O sistema deve funcionar numa máquina com GPU de consumo (mínimo 8 GB VRAM) sem dependência de hardware especializado.
RNF 7	Manutenção	A documentação do SIGP deve poder ser atualizada sem alterações ao código, bastando adicionar ou editar ficheiros <i>Markdown</i> e reindexar o <i>vectorstore</i> .

4.3 User Stories

Os *user stories* complementam os requisitos funcionais ao descrever as necessidades do sistema na perspetiva dos utilizadores finais. Foram identificados dois perfis, o utilizador direto do chatbot (agricultor ou técnico agrícola) e o administrador do sistema. A Tabela 4 apresenta os *user stories* identificadas.

Tabela 4 - User Stories

Nº	Perfil	User Story
US01	Utilizador	Como utilizador, quero fazer perguntas em linguagem natural sobre o SIGP para obter ajuda sem contactar o suporte técnico.
US02	Utilizador	Como utilizador, quero consultar dados das minhas parcelas, práticas agrícolas e adega em linguagem natural, sem conhecer SQL.

US03	Utilizador	Como utilizador, quero fazer perguntas de seguimento sem repetir o contexto da pergunta anterior.
US04	Utilizador	Como utilizador, quero que o sistema indique quando não sabe responder, em vez de fornecer informação incorreta.
US05	Utilizador	Como utilizador, quero receber respostas em português de Portugal, com linguagem simples e sem termos técnicos.
US06	Utilizador	Como utilizador, quero poder indicar quando uma resposta está errada, para que o problema possa ser corrigido.
US07	Administrador	Como administrador, quero atualizar a documentação do chatbot, editar ficheiros <i>Markdown</i> e adicionar outras fontes sem alterar o código.
US08	Administrador	Como administrador, quero monitorizar as interações do chatbot para identificar respostas incorretas e oportunidades de melhoria.
US09	Administrador	Como administrador, quero garantir que nenhum dado operacional sai da infraestrutura da empresa.
US10	Administrador	Como administrador, quero adicionar novas bases de dados ao sistema sem reescrever a lógica existente.

4.4 Diagrama de Casos de Uso

A Figura 5 apresenta o diagrama de casos de uso do sistema. Foram identificados dois atores principais, o utilizador do SIGP (agricultor ou técnico agrícola) e o administrador do sistema.

O utilizador interage com o chatbot através de quatro casos de uso principais. O primeiro, e mais frequente, é a consulta de documentação, onde o utilizador coloca questões sobre as funcionalidades dos módulos do SIGP e recebe respostas baseadas na documentação indexada.

O segundo caso de uso é a consulta de dados operacionais, em que o utilizador faz perguntas sobre informação armazenada nas bases de dados, como o número de parcelas, tipos de rega ou práticas culturais registadas, e o sistema traduz a pergunta para SQL, executa a *query* e devolve o resultado em linguagem natural.

O terceiro caso de uso corresponde às consultas mistas, que requerem simultaneamente informação da documentação e da base de dados. Nestas situações, o sistema executa ambos os *pipelines* em paralelo e agrega as respostas. Por fim, o utilizador pode também interagir de forma informal, com saudações ou mensagens de conversa, às quais o sistema responde de forma adequada sem invocar os pipelines de processamento.

O administrador do sistema tem dois casos de uso: a gestão da documentação, que consiste em adicionar ou atualizar os ficheiros *Markdown* que alimentam o *pipeline* RAG, e a monitorização do sistema através da plataforma Langfuse, onde pode analisar as interações, os tempos de resposta e o comportamento do modelo para identificar problemas ou oportunidades de melhoria.

A Figura 3 sintetiza os casos de uso descritos acima.

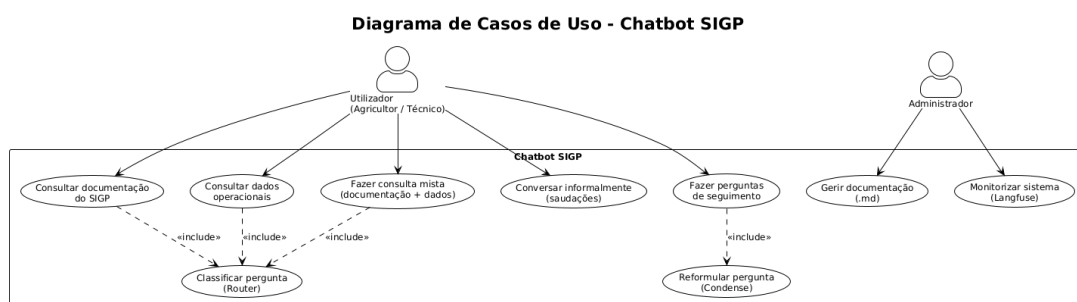


Figura 5 - Diagrama de Casos de Uso

4.5 Conceção da Solução

Esta secção apresenta a arquitetura do sistema desenvolvido e os seus principais componentes. O *chatbot* foi concebido como um sistema modular que combina duas abordagens complementares de inteligência artificial, *Retrieval-Augmented Generation* (RAG) para consulta de documentação e um agente SQL para consulta de bases de dados em linguagem natural.

4.5.1 Arquitetura Geral

A Figura 6 apresenta a arquitetura final do sistema.

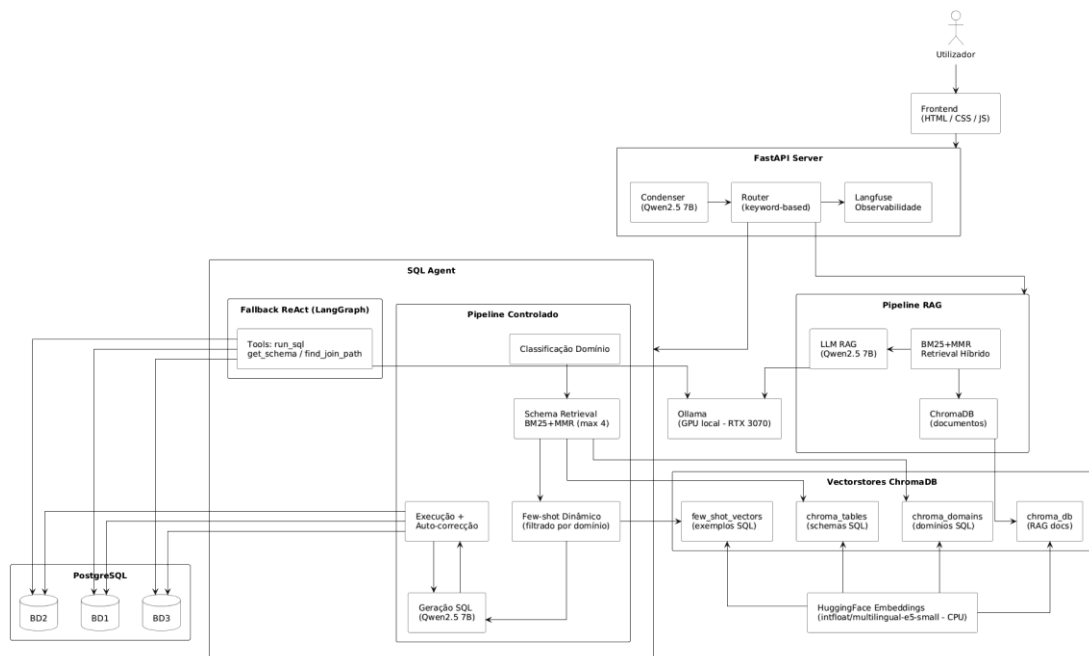


Figura 6 - Arquitetura do Sistema

O utilizador interage com o *chatbot* através de uma interface web. Cada pergunta é enviada ao servidor FastAPI, que a processa através de um pipeline com os seguintes

componentes, cujo fluxo de interação completo é ilustrado na Figura 7.



Figura 7 - Fluxo de Interação

O *router* é o primeiro componente a atuar. Analisa a pergunta recebida e classifica-a numa de seis categorias: RAG (documentação), SQL (bases de dados), AMBOS (documentação e dados em simultâneo), SAUDAÇÃO, CONVERSA ou META. Esta

classificação é feita exclusivamente por correspondência de palavras-chave, sem recurso a modelos de linguagem, o que a torna determinística e instantânea.

Quando existe um histórico de sessão, a pergunta do utilizador é primeiro reformulada por um modelo condensador que incorpora o contexto das mensagens anteriores, transformando perguntas dependentes de contexto (como "e qual a mínima?") em perguntas autónomas (como "qual a área mínima das parcelas?"). Este passo é essencial para que o router e os pipelines subsequentes possam interpretar corretamente a intenção do utilizador.

Consoante a classificação, a pergunta é encaminhada para o *pipeline* RAG, para o SQL Agent, ou para ambos em paralelo. No caso de AMBOS, as respostas parciais são agregadas num contexto único que é depois processado pelo modelo de linguagem para produzir uma resposta coerente.

4.5.2 Pipeline RAG

O *pipeline* RAG implementa o padrão proposto por (Lewis et al., 2020), que combina pesquisa de informação com geração de texto. A documentação do SIGP, escrita em ficheiros Markdown, é dividida em segmentos (*chunks*) e indexada numa base de dados vetorial (ChromaDB) utilizando *embeddings* multilingues. Quando o utilizador faz uma pergunta classificada como RAG, o sistema pesquisa os segmentos semanticamente mais relevantes e fornece-os como contexto ao modelo de linguagem, que gera a resposta com base exclusivamente nessa informação.

Esta abordagem evita que o modelo alucine informação não existente na documentação, uma vez que o *prompt* instrui explicitamente o modelo a responder apenas com base no contexto fornecido e a declarar quando não tem informação suficiente.

4.5.3 SQL Agent

O SQL Agent permite ao utilizador consultar as bases de dados PostgreSQL do SIGP em linguagem natural. Implementa o padrão ReAct (Yao et al., 2022), no qual o modelo de linguagem raciocina sobre a pergunta e decide que ferramentas invocar a cada passo. O agente dispõe de cinco ferramentas: pesquisa de tabelas relevantes, consulta de *schema*, amostra de dados, cálculo de caminhos de JOIN entre tabelas e execução de *queries* SQL.

A Figura 8 ilustra o ciclo do ReAct do SQL Agent, o ciclo típico de uma consulta envolve a identificação das tabelas relevantes, a inspeção do seu *schema* para conhecer as colunas e chaves estrangeiras, e a construção e execução da *query* SQL. O resultado é devolvido ao modelo, que o interpreta e formula uma resposta em linguagem natural. Este processo é iterativo, já que se a *query* falhar ou devolver resultados inesperados, o agente pode corrigir a abordagem e tentar novamente.

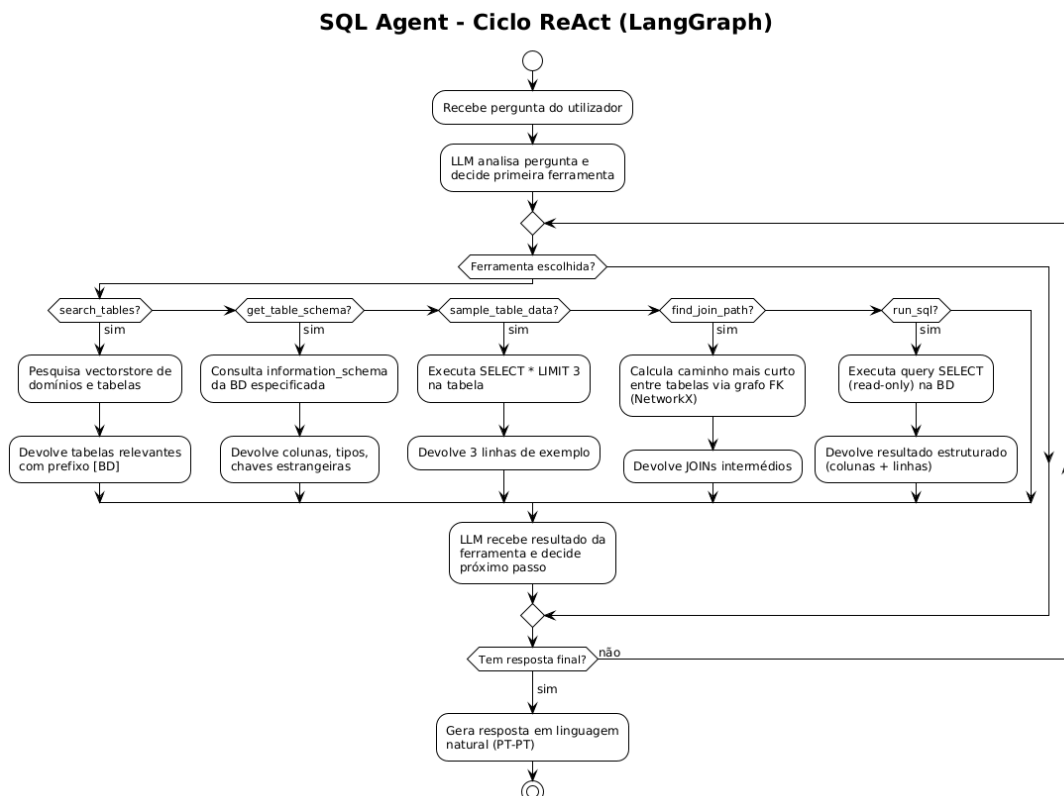


Figura 8 - Ciclo ReAct do SQL Agent

4.5.4 Observabilidade

Para monitorizar e diagnosticar o comportamento do sistema em tempo de execução, foi integrada a plataforma Langfuse (Langfuse, 2026), instalada localmente via Docker (Docker, 2025) para cumprir os requisitos de privacidade de dados da empresa. Esta ferramenta regista automaticamente cada interação do *chatbot*, incluindo os *prompts* enviados ao modelo, as respostas geradas, as ferramentas invocadas, os *tokens* consumidos e as latências de cada componente, permitindo assim a análise detalhada do *pipeline* de ponta a ponta. A análise destes dados foi determinante para identificar e corrigir os problemas de desempenho descritos nas secções seguintes.

4.6 Resumo

Em resumo, este capítulo apresenta a metodologia e conceção do sistema. A metodologia adotada baseia-se nos princípios ágeis, com desenvolvimento iterativo em ciclos curtos e acompanhamento semanal com os orientadores. São definidos os requisitos funcionais e não funcionais do chatbot, com destaque para o requisito de privacidade que condicionou a escolha de modelos locais. A conceção do sistema descreve a arquitetura modular, composta por um *router* de classificação, um *pipeline* RAG para consulta de documentação e um SQL Agent baseado no padrão ReAct para consulta de bases de dados, com observabilidade integrada via Langfuse.

5 Atividades Desenvolvidas

O presente capítulo descreve o trabalho realizado ao longo das seis fases do estágio, seguindo a estrutura do calendário de atividades, na Figura 3, acordado com a empresa. Para cada fase, apresentam-se as atividades realizadas, as decisões técnicas tomadas e os desvios ao plano inicial, devidamente justificados.

5.1 Fase 1: Análise e Preparação

A primeira fase teve como objetivo construir a base de conhecimento necessária para o desenvolvimento do sistema, através do estudo da plataforma SIGP e de uma revisão bibliográfica sobre as tecnologias a utilizar.

A primeira semana foi dedicada ao estudo aprofundado da plataforma SIGP, com acesso à documentação disponível no SharePoint da empresa. Este estudo abrangeu os principais módulos da plataforma: monitorização de parcelas e propriedades, práticas agrícolas, gestão de adegas entre outros.

Em paralelo, foi realizada uma revisão bibliográfica sobre as tecnologias a utilizar, nomeadamente os paradigmas de *Retrieval-Augmented Generation* e os modelos de *Large Language Models*, com foco em soluções locais que não impliquem o envio de dados para serviços externos, um requisito estabelecido pela empresa desde o início. Esta revisão fundamentou as decisões arquiteturais tomadas nas fases seguintes.

5.2 Fase 2: Configuração do Ambiente de Desenvolvimento

A segunda fase teve como objetivo preparar o ambiente de desenvolvimento e construir o *pipeline* de ingestão de documentos, que constituiu a base do sistema RAG.

5.2.1 Configuração do Ambiente de Desenvolvimento

A configuração do ambiente de desenvolvimento incluiu a instalação e configuração das principais ferramentas e bibliotecas: Python, FastAPI, LangChain, ChromaDB e o servidor de modelos locais Ollama. Uma decisão relevante nesta fase foi a escolha do modelo de linguagem. O plano inicial previa o Llama3, mas após avaliação comparativa, e como descrito na secção 3.1.5, optou-se pelo Qwen2.5 7B, que foi, também, escolhido pela empresa. O modelo selecionado demonstrou melhor desempenho em português e maior capacidade de seguir instruções complexas, requisito crítico para o *prompt engineering* do SQL Agent.

Para os *embeddings*, foi selecionado o modelo intfloat/multilingual-e5-small da HuggingFace, executado localmente em CPU. A escolha de um modelo multilingue foi determinada pela necessidade de indexar documentação em português e responder a perguntas com variações ortográficas e acentuação variável.

5.2.2 Pipeline de Ingestão de Documentos

O pipeline de ingestão é responsável por transformar os documentos *Markdown* da documentação do SIGP num *vectorstore*. O processo divide-se em três etapas:

Na primeira etapa, carregamento, os ficheiros *.md* são lidos da pasta de documentação. Os ficheiros com prefixo “_” são excluídos da indexação, de forma a permitir manter documentos de trabalho na mesma pasta sem contaminar o índice RAG.

Na segunda etapa, divisão em segmentos (*chunking*), cada documento é partido em blocos de texto com uma dimensão controlável. Cada segmento é prefixado com os metadados do módulo e da secção no formato [Módulo: X] [Secção: Y], o que permite ao LLM contextualizar a origem da informação sem expor caminhos de ficheiros ou a estrutura interna dos documentos.

Na terceira etapa, geração e armazenamento de *embeddings*, cada segmento é convertido numa representação vetorial e armazenado no ChromaDB, na pasta “chroma_db/”. O *vectorstore* é reconstruído automaticamente no arranque do servidor caso a pasta não exista, um mecanismo que simplifica a atualização da documentação sem alterações ao código.

5.3 Fase 3: Desenvolvimento do Core

A terceira fase foi a mais complexa do estágio, concentrando-se no desenvolvimento dos componentes centrais do sistema: o *pipeline* RAG, a API FastAPI, o *router* de classificação e o SQL Agent. Foi também nesta fase que as bases de dados foram disponibilizadas, o que permitiu iniciar o desenvolvimento da componente SQL.

5.3.1 Levantamento das Bases de Dados

Foram identificadas três bases de dados, cada uma com um domínio funcional distinto. A primeira, com cerca de 87 tabelas, contém informação sobre parcelas, propriedades, fitofármacos e dados geográficos. A segunda, com cerca de 167 tabelas, regista práticas culturais, recursos humanos, equipamentos, compras e rotas de trabalho. A terceira diz respeito ao módulo da adega, com aproximadamente 250 tabelas, suporta a gestão completa do processo produtivo de vinho.

Uma descoberta relevante foi a existência de tabelas com o mesmo nome em múltiplas bases de dados. Por exemplo, tabelas de parcelas, concelhos e tipos de rega existem simultaneamente na primeira e na segunda base de dados, com estruturas e dados parcialmente diferentes. A terceira base de dados, dedicada à gestão do processo produtivo, partilha também algumas tabelas de referência com as restantes,

nomeadamente variedades de casta e anos agrícolas. Esta análise condicionou diretamente as regras do *system prompt* do SQL Agent, que especifica explicitamente qual a base de dados a usar em cada caso de ambiguidade.

5.3.2 Sistema RAG

O pipeline RAG foi implementado com base no padrão *ConversationalRetrievalChain* do LangChain, que combina *retrieval* com memória de sessão. O *retriever* é um *ensemble* (LangChain, 2026) que combina duas estratégias complementares. A primeira é o MMR (*Maximal Marginal Relevance*) sobre *embeddings* densos, que assegura relevância semântica e diversidade dos *chunks*. A segunda é o BM25 (Gao et al., 2024), que garante bom desempenho em termos técnicos e nomes próprios do domínio SIGP (*fitofármaco, vinificação, NDVI*) onde os *embeddings* multilingues de menor dimensão tendem a ser menos bons. A fusão dos resultados segue *Reciprocal Rank Fusion* (Cormack et al., 2009) com pesos iguais, justificados empiricamente na Secção 5.4.3.

A gestão de contexto conversacional é assegurada por dois componentes: um modelo condensador, o Qwen2.5 7B, que reformula perguntas de seguimento em perguntas autónomas, incorporando o histórico das últimas três mensagens; e um dicionário de sessões que mantém o histórico por “*session_id*”.

Foi implementado um mecanismo de *fallback* no condensador, ou seja, se a pergunta reformulada for mais de três vezes maior que a original, o sistema usa a pergunta original de forma a evitar contaminações do histórico em modelos de linguagem de menor capacidade.

O QA (*Question Answering*) *prompt* instrui explicitamente o modelo a responder apenas com base no contexto fornecido e a declarar quando não tem informação suficiente, em vez de gerar informação incorreta. Esta regra revelou-se crítica durante os testes, já que, sem ela, o modelo tendia a alucinar passos de navegação a partir dos prefixos de secção dos documentos. A Figura 9 apresenta a construção da cadeia RAG, incluindo a definição do *retriever* híbrido e a ligação dos componentes de memória e *prompt*.

```
def build_rag_chain(vectorstore, callbacks=None):
    llm = OllamaLLM(model=LLM_MODEL, temperature=LLM_TEMPERATURE,
                    num_ctx=LLM_NUM_CTX)

    # MMR
    mmr = vectorstore.as_retriever(
        search_type="mmr",
        search_kwargs={"k": 5, "fetch_k": 20, "lambda_mult": 0.7},
    )

    # BM25
    raw = vectorstore.get(include=["documents", "metadatas"])
    bm25_docs = [Document(page_content=t, metadata=m or {})
                  for t, m in zip(raw["documents"], raw["metadatas"])]
    bm25 = BM25Retriever.from_documents(bm25_docs)
    bm25.k = 5

    # Fusão dos dois retrievers
    retriever = EnsembleRetriever(
        retrievers=[bm25, mmr],
        weights=[HYBRID_BM25_WEIGHT, HYBRID_MMR_WEIGHT],
    )
    ...
```

Figura 9 - Construção do retriever híbrido

A inclusão do BM25 resolve o problema da perda de vocabulário técnico (como fitofármaco ou trasfega). O modelo multilingual-e5-small tem dificuldade em mapear o significado exato dos termos específicos do domínio agrícola, o que gera representações vetoriais difusas para estes termos. O BM25 atua como um mecanismo de segurança com correspondência exata, enquanto o MMR interpreta a intenção semântica da pergunta e garante a diversidade da informação recuperada.

5.3.3 API FastAPI

A API foi desenvolvida com FastAPI, com um *endpoint* principal “/chat” que recebe a pergunta do utilizador e o identificador de sessão, devolvendo a resposta, a categoria classificada pelo router e o tempo de resposta. O servidor suporta respostas em *streaming*, transmitindo os *tokens* gerados progressivamente para o *frontend* à medida

que são produzidos pelo modelo, o que melhora a percepção de velocidade por parte do utilizador.

A gestão de sessões é feita através de um dicionário em memória indexado por um identificador de sessão, “session_id”, com histórico limitado a 3 mensagens. Esta restrição serve para controlar o crescimento do contexto e reduzir a latência. O servidor arranca com um mecanismo de ciclo de vida que inicializa todos os componentes, bases de dados vetoriais, modelos de linguagem e agentes antes de aceitar pedidos, de forma a garantir que a primeira resposta não sofra de uma sobrecarga computacional no arranque.

5.3.4 Router de Classificação

O router foi implementado como um classificador baseado em palavras-chave, sem recurso a modelos de linguagem. Esta decisão, divergente do plano inicial que previa um modelo Qwen2.5 0.5B para a classificação, foi tomada após constatar que um LLM introduzia latência e imprevisibilidade desnecessárias numa tarefa determinística. O módulo analisa cada mensagem do utilizador e classifica-a em seis categorias, RAG (documentação), SQL (base de dados), AMBOS, SAUDAÇÃO, CONVERSA e META, o que garante assim uma latência inferior a 1 milissegundo. A Figura 10 apresenta a lógica principal do classificador.

```
def classify(question: str) -> str:
    q = normalize(question.strip())
    if q in {normalize(g) for g in GREETINGS}:
        return "GREETING"
    if q in {normalize(c) for c in CHITCHAT}:
        return "CHITCHAT"
    if any(normalize(p) in q for p in META_PATTERNS):
        return "META"

    has_rag    = any(normalize(kw) in q for kw in RAG_ONLY)
    has_strong = any(normalize(kw) in q for kw in SQL_STRONG)
    has_weak   = any(normalize(kw) in q for kw in SQL_WEAK)
    has_entity = any(normalize(ent) in q for ent in DB_ENTITIES)

    if has_rag:
```

```
        return "BOTH" if (has_strong and has_entity) else "RAG"
    if (has_strong or has_weak) and has_entity:
        return "SQL"
    if has_entity:
        return "SQL"
    return "RAG"
```

Figura 10 - Função de classificação do router

A lógica de classificação distingue entre palavras-chave SQL fortes (quantificadores como "quantas", "total", "soma", "média") e fracas (verbos como "lista", "mostra", "quais"). Uma pergunta só é classificada como SQL se contiver uma palavra-chave forte ou uma palavra-chave fraca acompanhada de uma entidade das bases de dados. Esta distinção evita que perguntas como "mostra-me como funciona a adega" sejam incorretamente encaminhadas para o SQL Agent. A categoria AMBOS é atribuída apenas quando existem simultaneamente indicadores fortes de RAG e de SQL, o que previne a penalização desnecessária de latência em perguntas ambíguas.

Por fim, a categoria META destina-se a identificar questões sobre a identidade, as capacidades e o âmbito de atuação do próprio *chatbot*, como pedidos de listagem dos módulos ou funcionalidades suportadas, o que permite ao sistema esclarecer o seu propósito de forma direta, sem necessidade de acionar os motores de pesquisa RAG ou SQL.

O router foi validado com 220 testes unitários, de forma a cobrir todas as categorias tal como casos limite como frases com acentos, maiúsculas, abreviaturas e perguntas ambíguas.

5.3.5 SQL Agent

O SQL Agent é implementado como um agente ReAct utilizando a *framework* LangGraph. Ao contrário de uma abordagem *text-to-SQL* tradicional, o agente dispõe de cinco ferramentas que pode invocar iterativamente para explorar o esquema, validar hipóteses e executar consultas. Esta abordagem é essencial num ambiente com três bases de dados e mais de 500 tabelas, onde é impossível incluir todo o

esquema no contexto do LLM. A Figura 11 apresenta as ferramentas utilizadas pelo agente SQL.

```
tools_list = [
    search_tables,      # pesquisa vectorial de tabelas por domínio
    get_table_schema,   # colunas, tipos e chaves estrangeiras
    sample_table_data,  # 3 linhas de amostra para perceber formatos
    run_sql,            # execução de SELECT com fallback multi-BD
    find_join_path,     # cálculo automático de JOIN via grafo FK
]

self.agent = create_agent(
    model=llm,
    tools=tools_list,
    system_prompt=SYSTEM_PROMPT,
)
```

Figura 11 - Definição das cinco ferramentas disponíveis ao agente SQL

A ferramenta `find_join_path` merece ser destacada pois constrói um grafo dirigido a partir das chaves estrangeiras de cada base de dados com a biblioteca `NetworkX` (*NetworkX*, 2026), e calcula o caminho mais curto entre duas tabelas arbitrárias. Isto evita que o agente tenha de descobrir os JOIN por tentativa e erro, reduzindo drasticamente o número de iterações em perguntas que envolvem múltiplas tabelas.

O system prompt do SQL Agent inclui regras explícitas para garantir a qualidade das respostas ao utilizador final. Em primeiro lugar, as respostas devem apresentar o campo de denominação ou nome em vez de identificadores numéricos internos, uma vez que IDs como "42" não têm significado para o utilizador. Em segundo lugar, campos de gestão interna como `gid`, `usercreate`, `datacreate` e `userupdate` são excluídos dos resultados. Por fim, todos os filtros WHERE devem incluir a condição estado NOT IN ('Y', 'Z') para excluir registos inativos e eliminados, garantindo que o utilizador vê apenas dados operacionais ativos.

5.4 Fase 4: Interface e Observabilidade

A quarta fase centrou-se em três frentes paralelas, o desenvolvimento da interface web, a integração do sistema de observabilidade Langfuse e a realização de otimizações de desempenho motivadas pelos problemas identificados nos primeiros testes de utilização.

5.4.1 Interface Web

A interface web foi desenvolvida com HTML, CSS e JavaScript puros, sem dependências de frameworks de frontend, de forma a minimizar a complexidade de *deployment* e manter a *stack* tecnológica simples. A interface oferece uma janela de *chat* com suporte a *streaming* de *tokens*, que transmite a resposta progressivamente à medida que o modelo a gera, melhorando assim a perceção de velocidade por parte do utilizador. Cada resposta é acompanhada do tempo de resposta, visível abaixo de cada mensagem do assistente.

A comunicação entre a interface de utilizador e o servidor processa-se através de ferramentas nativas dos navegadores web. A ligação ao ponto de acesso da API responsável pelo *chatbot* (o *endpoint/chat*) estabelece-se através de um fluxo contínuo de eventos. Esta arquitetura possibilita a receção e a apresentação dos *tokens* no ecrã em tempo real, à medida que o modelo de linguagem os processa.

Para gerir os diálogos, o sistema cria um identificador único de conversa no momento exato em que o utilizador abre a página. Este código fica guardado de forma temporária na memória isolada do separador. Esta abordagem assegura a total independência entre sessões e dita que uma simples atualização da página inicie um contexto de conversação completamente limpo. Esta lógica erradica, assim, o problema de contaminação cruzada do histórico de mensagens identificado durante a fase de testes. A Figura 12 ilustra o resultado desta interface web durante uma interação com o utilizador.

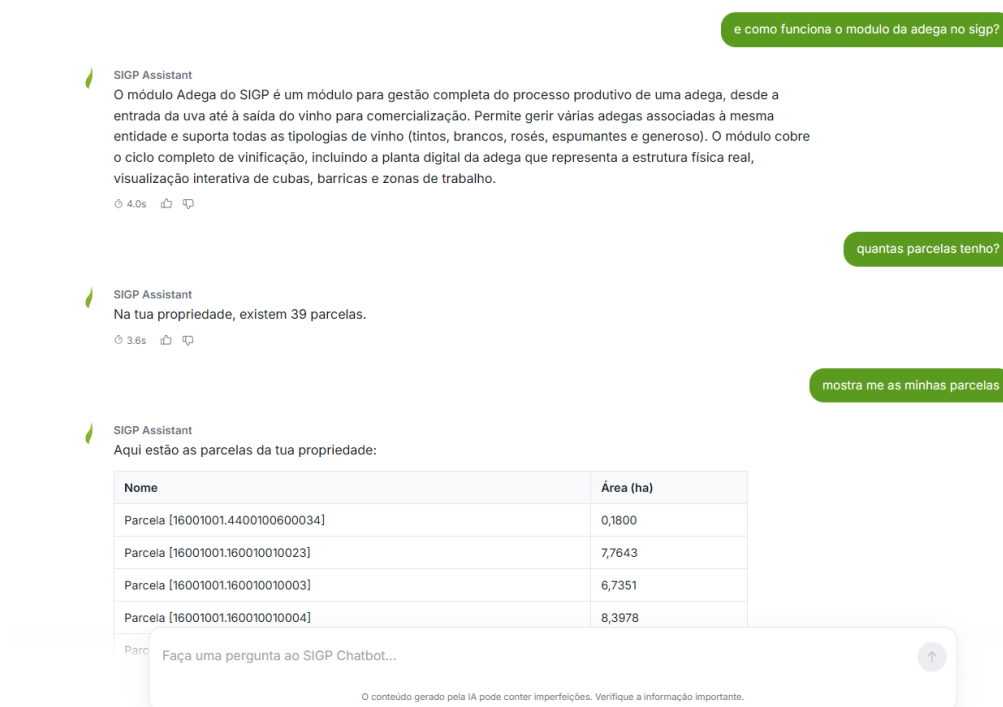


Figura 12 - Interface Web do Chatbot

A interface inclui também um ecrã de inicialização de sessão, apresentado antes do acesso ao chat, onde o utilizador seleciona a entidade e o ano agrícola. Estes valores ficam guardados em sessionStorage e são enviados em cada pedido ao servidor, o que permite ao SQL Agent filtrar automaticamente os dados pelo contexto da exploração sem que o utilizador precise de especificar esse contexto em cada pergunta.

A interface, como ilustrada na Figura 13, inclui um painel de administração, que permite gerir a base de conhecimento do chatbot sem acesso direto ao servidor. O painel está organizado em duas secções: o carregamento de documentos e a lista dos documentos indexados.

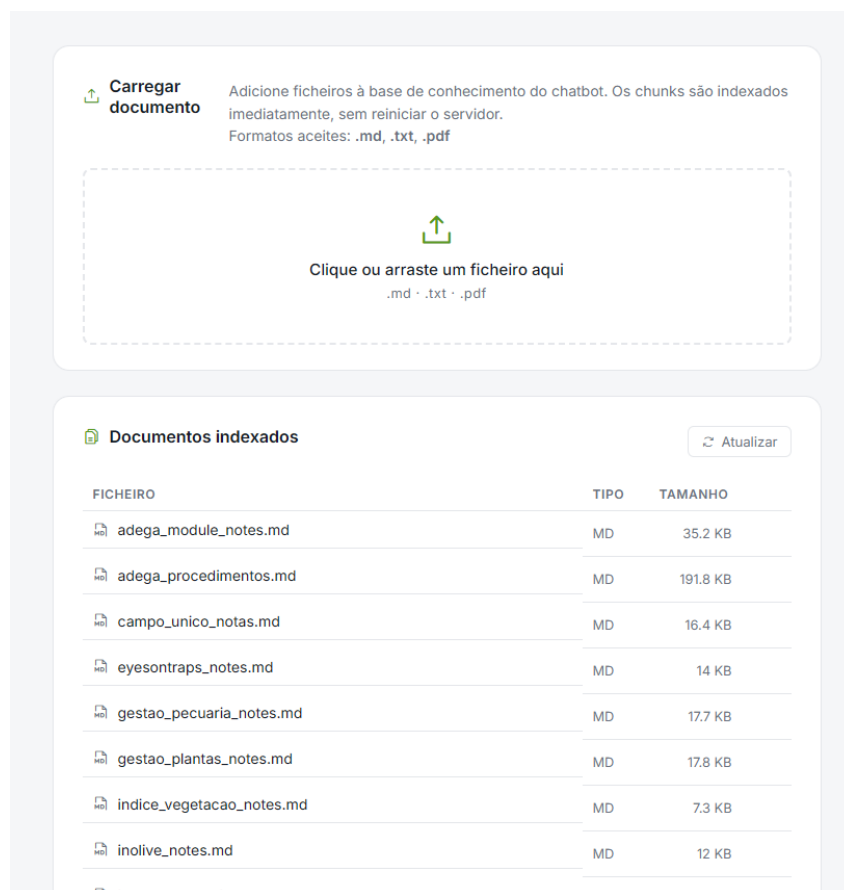


Figura 13 - Interface da página de administrador

O carregamento suporta ficheiros nos formatos .md, .txt e .pdf, e é realizado por clique ou por arrastar e largar. Após o envio, o documento é processado de imediato pelo pipeline de ingestão RAG, sendo que os novos *chunks* ficam disponíveis para pesquisa sem necessidade de reiniciar o servidor. Além disso, o número de blocos indexados é apresentado na interface.

A lista apresenta todos os documentos indexados, com o nome, tipo e tamanho. Cada documento tem um botão de eliminação que remove o ficheiro do disco e os respetivos *chunks* do *vectorstore* ChromaDB em tempo real, sem interrupção do serviço. Um botão de atualização permite sincronizar manualmente a lista sem recarregar a página.

5.4.2 Observabilidade com Langfuse

A integração do Langfuse em modo *self-hosted*, via Docker, foi realizada com o objetivo de registrar e analisar os *traces* de todas as chamadas ao sistema, sem que os dados saiam do servidor local. O Langfuse é uma plataforma de observabilidade para aplicações LLM que regista cada passo de execução, chamadas ao modelo, ferramentas invocadas, inputs e outputs numa interface web navegável.

A integração foi feita através do “CallbackHandler” do pacote “langfuse”, passado como *callback* às *chains* do LangChain e ao agente LangGraph. Desta forma, cada pedido ao endpoint “/chat” gera automaticamente um *trace* com todos os passos de execução, tempos de resposta por componente e *tokens* consumidos.

A observabilidade revelou-se um instrumento de desenvolvimento essencial, já que foi através da análise dos *traces* que foram identificados os problemas do SQL Agent descritos na secção 5.3.5, nomeadamente a alucinação após resultados vazios e o caminho de JOIN incorreto, bem como o problema de desempenho descrito na secção 5.4.3.

A Figura 14 apresenta o *dashboard* principal do Langfuse.

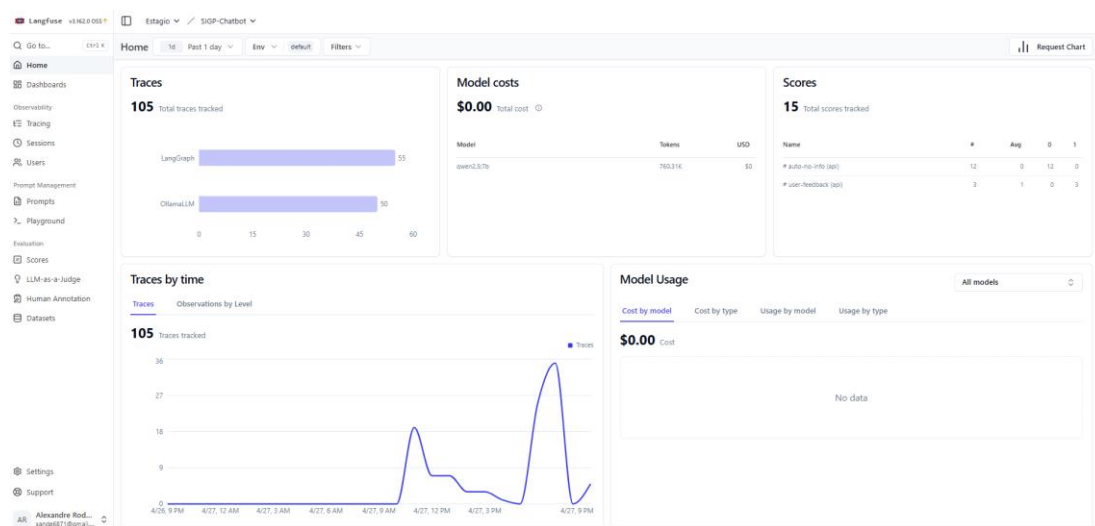


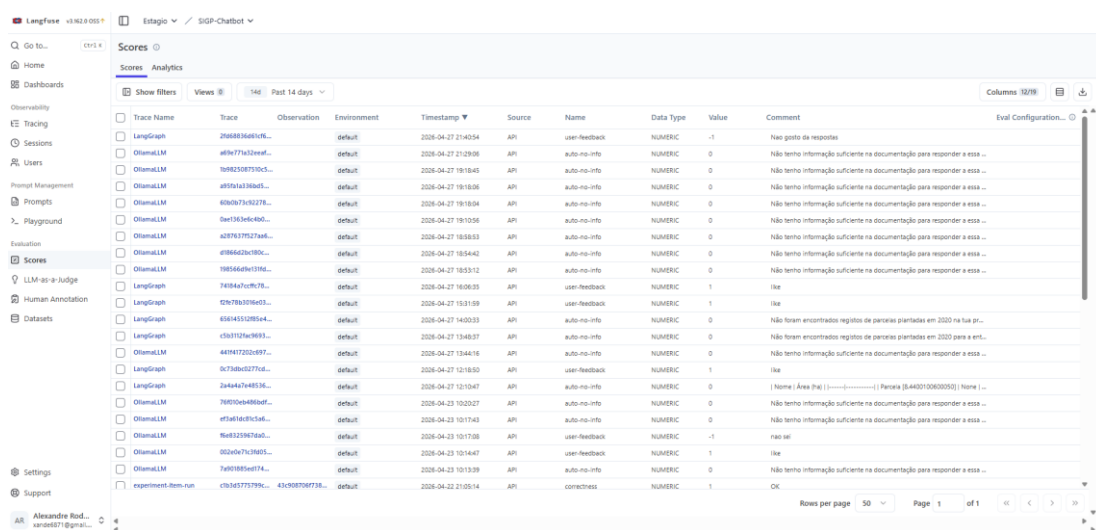
Figura 14 - Dashboard do Langfuse

Como é possível observar na Figura 14, o *dashboard* principal do Langfuse oferece uma visão agregada e em tempo real da saúde do sistema. Através desta interface, é

possível monitorizar métricas vitais como o volume total de *traces*, a evolução da latência média das respostas e o consumo global de *tokens*, que é 0 pois usamos um modelo local. Esta representação visual permite à equipa de desenvolvimento identificar rapidamente picos de tráfego, estrangulamentos de performance ou anomalias no comportamento do LLM sem necessidade de analisar os *logs* em texto bruto.

Para além do registo automático de *traces*, foi implementado um mecanismo de *feedback* de utilizador diretamente na interface web. Cada resposta do *chatbot* é acompanhada de botões de aprovação e rejeição. No caso de rejeição, o utilizador pode, opcionalmente, indicar o motivo num campo de texto livre. Este *feedback* é enviado ao Langfuse como um *score* (uma pontuação) do tipo “user-feedback” (valor +1 ou -1) associado ao *trace* correspondente, ficando assim visível na consola juntamente com o *input*, *output* e passos de execução da resposta rejeitada. Foi também implementada a deteção automática de respostas sem informação, sempre que o modelo responde com a frase “Não tenho informação suficiente”. O sistema envia automaticamente um *score* “auto-no-info” ao Langfuse, o que permite monitorizar as perguntas que o sistema não consegue responder com a documentação disponível.

A Figura 15 detalha o painel de scores do Langfuse.



Trace Name	Trace	Observation	Environment	Timestamp	Source	Name	Data Type	Value	Comment
LangGraph	2668366106...		default	2026-04-27 21:40:04	API	user-feedback	NUMERIC	-1	Isse gosto da resposta
OllamaLLM	af6b773a39eaf...		default	2026-04-27 19:18:45	API	auto-no-info	NUMERIC	0	Não tenho informação suficiente na documentação para responder a essa ...
OllamaLLM	7682508702c...		default	2026-04-27 19:18:06	API	auto-no-info	NUMERIC	0	Não tenho informação suficiente na documentação para responder a essa ...
OllamaLLM	af97a1a338aaf...		default	2026-04-27 19:18:04	API	auto-no-info	NUMERIC	0	Não tenho informação suficiente na documentação para responder a essa ...
OllamaLLM	636a073a92278...		default	2026-04-27 19:18:04	API	auto-no-info	NUMERIC	0	Não tenho informação suficiente na documentação para responder a essa ...
OllamaLLM	0ae19c3de46b...		default	2026-04-27 19:10:06	API	auto-no-info	NUMERIC	0	Não tenho informação suficiente na documentação para responder a essa ...
OllamaLLM	a287f37927aaf...		default	2026-04-27 18:58:53	API	auto-no-info	NUMERIC	0	Não tenho informação suficiente na documentação para responder a essa ...
OllamaLLM	d986623c380c...		default	2026-04-27 18:54:42	API	auto-no-info	NUMERIC	0	Não tenho informação suficiente na documentação para responder a essa ...
OllamaLLM	78856d9c319a...		default	2026-04-27 18:53:12	API	auto-no-info	NUMERIC	0	Não tenho informação suficiente na documentação para responder a essa ...
LangGraph	7404a7c09c7b...		default	2026-04-27 18:06:35	API	user-feedback	NUMERIC	1	Isse
LangGraph	02b78b3c01e43...		default	2026-04-27 15:31:59	API	user-feedback	NUMERIC	1	Isse
LangGraph	6844532081e4...		default	2026-04-27 14:00:53	API	auto-no-info	NUMERIC	0	Não foram encontrados registos de parâmetros criados em 2020 na tua pr...
LangGraph	c3b3123a0693...		default	2026-04-27 13:48:37	API	auto-no-info	NUMERIC	0	Não foram encontrados registos de parâmetros criados em 2020 para a ent...
OllamaLLM	44947232c897...		default	2026-04-27 13:44:16	API	auto-no-info	NUMERIC	0	Não tenho informação suficiente na documentação para responder a essa ...
LangGraph	0c73d6d2776a...		default	2026-04-27 12:18:50	API	user-feedback	NUMERIC	1	Isse
LangGraph	2a44a7a48136...		default	2026-04-27 12:10:47	API	auto-no-info	NUMERIC	0	Nome Área (ha) Período (B-440010060050) Nome ...
OllamaLLM	78010b4860af...		default	2026-04-23 10:20:27	API	auto-no-info	NUMERIC	0	Não tenho informação suficiente na documentação para responder a essa ...
OllamaLLM	ef3a16d181c4f...		default	2026-04-23 10:17:43	API	auto-no-info	NUMERIC	0	Não tenho informação suficiente na documentação para responder a essa ...
OllamaLLM	9a82259c7a0b...		default	2026-04-23 10:17:08	API	user-feedback	NUMERIC	-1	nao sei
OllamaLLM	022dc97c36a5...		default	2026-04-23 10:14:47	API	user-feedback	NUMERIC	1	Isse
OllamaLLM	7a01885aef174...		default	2026-04-23 10:13:39	API	auto-no-info	NUMERIC	0	Não tenho informação suficiente na documentação para responder a essa ...
experiment-item-run	c7b35757579c...		default	2026-04-22 21:05:14	API	correctness	NUMERIC	1	OK

Figura 15 - Scores do Langfuse, onde se pode ver a qualidade das respostas de acordo com os utilizadores

Como se observa na Figura 15, o feedback qualitativo é centralizado e quantificado. Na imagem, é visível a listagem das avaliações submetidas, permitindo distinguir rapidamente as interações bem-sucedidas das rejeitadas. A partir desta vista, o administrador do sistema pode selecionar qualquer avaliação negativa e inspecionar a árvore de execução exata, desde o *prompt* inicial até às ferramentas invocadas, e o que originou a insatisfação do utilizador, facilitando a correção do modelo e dos documentos RAG associados.

5.4.3 Otimizações de Desempenho

A análise dos *traces* Langfuse revelou que o tempo de resposta na primeira chamada após o arranque do servidor era anormalmente elevado, cerca de 33 segundos na componente de condensação da pergunta. A causa foi identificada como uma diferença no parâmetro “num_ctx” (tamanho da janela de contexto) entre os três modelos Ollama configurados: o modelo de condensação usava “num_ctx=768”, o RAG usava “num_ctx=4096” e o SQL usava “num_ctx=8192”. O Ollama recarrega o modelo quando este parâmetro muda entre chamadas consecutivas, o que causava uma pausa de aproximadamente 30 segundos em cada transição de componente.

A solução foi unificar o parâmetro “num_ctx=8192” para todos os modelos. Com esta alteração, o Ollama mantém o mesmo modelo carregado em memória GPU, independentemente do componente invocado. O impacto foi significativo, a latência do sistema estabilizou em 2–4 segundos por pergunta em regime de utilização normal, contra os 30–67 segundos observados antes da correção.

Uma segunda otimização foi a substituição do modelo de condensação de “qwen2.5:1.5b” para “qwen2.5:7b”, o mesmo modelo usado nos restantes componentes. Além de eliminar um recarregamento adicional, o modelo maior demonstrou maior fidelidade na reformulação de perguntas de seguimento, o que reduz a contaminação do histórico de sessão.

Uma terceira otimização incidiu sobre o *router* de classificação. Durante os testes funcionais, verificou-se que perguntas sobre funcionalidades do SIGP com terminologia ambígua (como “como funciona”, “o que é”, “como se regista”) eram por

vezes encaminhadas para o SQL Agent em vez do *pipeline* RAG, o que resultava em tempos de resposta elevados e respostas incorretas. A solução foi a definição de um conjunto de palavras-chave que forçam o RAG (FORCE_RAG), garantindo o encaminhamento para o *pipeline* documental independentemente da presença de entidades das bases de dados na pergunta. O impacto foi significativo, perguntas anteriormente com latências de 12 a 53 segundos passaram a ser resolvidas em 1 ou 2 segundos.

5.4.4 Melhorias Iterativas à Qualidade das Respostas

Ao longo dos testes de utilização, foram identificados e corrigidos quatro problemas com impacto direto na qualidade das respostas. A função “fixMarkdownLists” foi reescrita como um *parser* iterativo linha a linha, isto é, o texto é dividido em `\n`, e cada linha é classificada como tabela (início com `|`), lista (`-`, `*`, `+` ou `1.`) ou texto simples. Uma quebra de linha extra é injetada apenas quando uma nova tabela ou lista começa a seguir a texto. A versão anterior, baseada em expressões regulares, produzia formatação incorreta durante o *streaming* SSE porque os *chunks* de texto chegavam incompletos e ativavam substituições parciais. Em paralelo, a limpeza de espaços no *backend* foi corrigida: a expressão “`\s{2,}`” foi substituída por “`[ \t]{2,}`”, preservando, assim, as quebras de linha nas tabelas e parágrafos antes de chegarem ao *renderer* do frontend.

Foi implementada a função “compress_history” no *endpoint* `/chat/stream`, que interceta as mensagens do assistente antes de as passar ao SQL Agent e substitui tabelas *Markdown* no histórico por um marcador textual que instrui o agente a executar sempre uma nova *query* SQL em vez de reutilizar dados anteriores. Este mecanismo eliminou um padrão de alucinação recorrente em perguntas de seguimento, em que o agente reproduzia os dados já presentes no histórico sem aceder à base de dados, produzindo resultados incorretos quando o utilizador aplicava novos filtros. Como ilustrado na Figura 16, este mecanismo interceta as mensagens do assistente antes da sua passagem para o agente SQL e substitui as tabelas em formato *Markdown* por um marcador textual descritivo. Esta instrução

obriga o agente a executar uma nova consulta SQL em vez de reutilizar dados prévios da memória da sessão.

```
def _compress_history(messages):
    compressed = []
    for msg in messages:
        if msg.type == "ai" and "|" in msg.content and "---" in msg.content:
            row_count = len(msg.content.split("\n")) - 2
            resumo = (
                f"[Forneci os dados pedidos numa tabela com {max(0, row_count)} linhas. "
                f"Não tenho estes dados em memória. Para novos filtros, "
                f"SOU OBRIGADO a executar nova query SQL.]"
            )
            compressed.append(AIMessage(content=resumo))
        else:
            compressed.append(msg)
    return compressed
```

Figura 16 - Função `_compress_history`: substituição de tabelas no histórico para forçar nova consulta SQL

Foram adicionadas ao *system prompt* do SQL Agent regras explícitas para os problemas de *schema* identificados nos traces do Langfuse. Por exemplo, a coluna `ano_plantacao` é do tipo `NUMERIC` e não suporta `EXTRACT()` em SQL; a tabela `parcela` não tem uma coluna `id_entidade`; ordenações com `ORDER BY ... DESC` devem incluir `NULLS LAST` para evitar que valores nulos apareçam no topo dos resultados. A função `"run_sql"` foi também alterada para converter valores `"None"` do PostgreSQL em `"N/A"` antes de devolver o resultado ao modelo, bem como para incluir uma dica de sistema nos erros de SQL para instruir o agente a consultar o *schema* antes de reformular a *query*.

Por último, a função `"_condense_question"`, responsável por reescrever a pergunta do utilizador com o histórico da sessão, foi movida para o início do *endpoint*, antes da chamada ao router de classificação. Na versão anterior, os *follow-ups* curtos como "E quais foram plantadas em 2020?" chegavam ao router sem contexto, eram classificados incorretamente e enviados para o componente errado. Com a reordenação, a pergunta chega ao router e ao SQL Agent já expandida com o contexto necessário.

5.4.5 Otimização Avançada do Pipeline SQL

A versão inicial do SQL Agent, descrita na secção 5.3.5, assentava exclusivamente num agente ReAct com acesso livre a ferramentas de inspeção de esquemas e execução de SQL. Embora funcional para perguntas simples, esta abordagem revelou limitações sistemáticas em produção, o agente entrava com frequência em ciclos de tentativa e erro que esgotavam o limite de passos configurado, seleccionava tabelas irrelevantes por falta de contexto semântico suficiente, e produzia SQL com nomes de colunas incorretos por não verificar o esquema antes de gerar a *query*. O resultado era uma taxa de acerto de 13% no *golden set* da GeoDouro.

Para resolver estes problemas, o SQL Agent foi reestruturado em torno de um pipeline determinístico de cinco etapas que substitui o ReAct como caminho principal, mantendo-o apenas como mecanismo de recurso para perguntas que o pipeline não consiga resolver. A Figura 17 ilustra o fluxo completo do sistema otimizado.

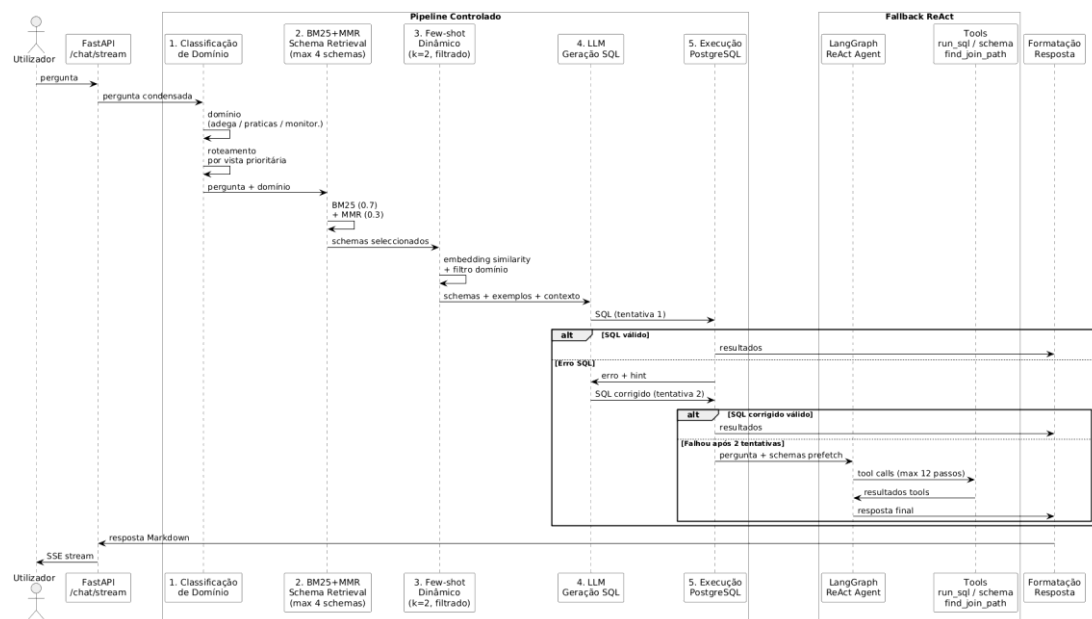


Figura 17 - Diagrama do pipeline SQL otimizado

Na primeira etapa, a pergunta é classificada por domínio (monitorização, práticas agrícolas ou adega) e por tipo de consulta, o que permite ativar um roteamento prioritário para vistas específicas antes de qualquer pesquisa. Na segunda etapa, os

esquemas das tabelas mais relevantes são recuperados através de pesquisa híbrida BM25+MMR tal como no RAG, com um limite máximo de quatro esquemas injetados no contexto. Na terceira etapa, exemplos de pares pergunta-SQL semanticamente semelhantes são recuperados de uma base de exemplos anotados e injetados no *prompt*. Na quarta etapa, o modelo gera a instrução SQL com base no contexto enriquecido. Na quinta etapa, a *query* é executada e, em caso de erro, o modelo tem uma segunda tentativa com a mensagem de erro como contexto adicional. Se ambas as tentativas falharem, o controle passa ao agente ReAct.

5.4.6 Pesquisa Híbrida BM25+MMR e Schema Pruning

A seleção das tabelas relevantes para uma dada pergunta é determinante para a qualidade do SQL gerado. Note-se que um esquema desnecessário ocupa contexto e aumenta a probabilidade de erro, enquanto a omissão de uma tabela indispensável inviabiliza a resposta. A versão inicial usava apenas pesquisa por similaridade vetorial, que falhava em termos exatos de denominação de recursos como nomes de fitofármacos ou lotes de vinho. A solução adotada combina pesquisa BM25, que privilegia a correspondência lexical exata, com reclassificação por MMR, que diversifica os resultados e evita a recuperação de tabelas redundantes. Os dois scores são fundidos com pesos de 70% para BM25 e 30% para MMR, determinados empiricamente sobre o conjunto de validação. O número máximo de esquemas injetados no *prompt* foi fixado em quatro, limite a partir do qual o modelo começa a ignorar esquemas em posições distantes no contexto, fenómeno conhecido como *lost-in-the-middle*.

5.4.7 Demonstrações de Poucas Amostras Dinâmicas

O *prompt* do pipeline inclui exemplos de pares pergunta-SQL para orientar a geração. Na versão estática, os exemplos eram fixos e cobriam apenas os padrões mais frequentes das práticas agrícolas, o que produzia erros nas perguntas sobre outros temas. A versão final recupera dinamicamente os exemplos mais semelhantes à pergunta atual por similaridade de *embeddings*, a partir de um documento com trinta

exemplos anotados. A recuperação é filtrada por domínio, sendo que, por exemplo, perguntas sobre adegas recebem exclusivamente exemplos de adegas, permitindo a eliminação da contaminação cruzada entre domínios com padrões SQL distintos.

5.4.8 Vistas Materializadas

As perguntas de agregação sobre práticas culturais exigiam a execução de múltiplos JOIN em tempo real sobre dezenas de milhares de registros, o que resultava sistematicamente em *timeout*. Foram criadas quatro vistas materializadas na base de dados que pré-calculam os dados de agregação mais consultados e os que persistem como tabelas físicas indexadas. Com estas vistas, as perguntas que antes resultavam em *timeout* passaram a obter resposta em menos de cinco segundos pelo pipeline. A contrapartida é que os dados requerem atualização manual após ingestão de novos registros.

5.5 Fase 5: Testes

A quinta fase foi dedicada à validação do sistema através de testes estruturados, com cobertura progressiva desde os componentes mais determinísticos até aos fluxos de integração completos.

5.5.1 Testes Unitários ao Router

O router de classificação foi o primeiro componente a ser coberto por testes unitários, por ser determinístico e não depender de modelos de linguagem. Foram desenvolvidos 220 testes, organizados por categoria de classificação (RAG, SQL, AMBOS, SAUDAÇÃO, CONVERSA), com cobertura explícita de casos limite como frases com acentos, maiúsculas, abreviaturas e entidades das bases de dados em contextos conversacionais. A suite é executada sem dependências externas e serve como rede de segurança para futuras alterações à lógica de classificação.

5.5.2 Testes de Integração SQL

Os testes de integração ao SQL Agent foram realizados com perguntas de referência distribuídas por quatro classes de complexidade, *queries* de tabela única, *queries* com JOIN entre duas tabelas, *queries* com agregações (COUNT, SUM, AVG) e *queries* que exigem navegação por múltiplas tabelas via “find_join_path”. Para cada pergunta, a resposta esperada foi previamente verificada por consulta direta à base de dados, o que permitiu comparar o resultado do agente com o valor de referência.

Os testes de integração revelaram dois padrões de falha recorrentes. O primeiro consistia no agente invocar “run_sql” sem ter verificado previamente o *schema* da tabela, o que resultava em *queries* com nomes de colunas incorretos. Em segundo, perguntas que envolviam tabelas presentes em múltiplas bases de dados, o agente escolhia a base de dados incorreta sem consultar as regras do *system prompt*. Ambos os problemas foram corrigidos com refinamentos adicionais ao *system prompt* e, no segundo caso, com a adição de uma regra explícita de desambiguação por domínio.

5.5.3 Testes Funcionais

Para avaliar a qualidade do sistema foram construídos três conjuntos de testes, 220 testes unitários do router, 50 perguntas do *golden set* de RAG e 100 perguntas do *golden set* de SQL. Todas as perguntas têm resposta esperada determinística (*keyword matching*) e cobrem as três bases de dados e os três módulos documentais. A Tabela 5 resume os resultados.

Os testes funcionais foram realizados com recurso a um conjunto de perguntas representativas do uso real do sistema, cobrindo os principais cenários de utilização, consultas de documentação, consultas de dados operacionais, consultas mistas e perguntas de seguimento. Cada resposta foi avaliada quanto à correção do conteúdo, ao componente invocado pelo *router* e à ausência de alucinações. Os testes funcionais serviram também para validar a percepção de velocidade da interface com *streaming* ativo.

Tabela 5 - Resultados dos *benchmarks* automatizados

Componente	Total	Acerto	Latência média
Router	220	100%	< 1ms
RAG	50	100%	3.3s
SQL Agent	100	100%	3.9s

Os resultados apresentados na Tabela 5 validam o sistema nos três componentes testados. O router obteve 100% de exatidão em 220 casos de teste com latência inferior a 1 milissegundo, o que confirma que a abordagem baseada em palavras-chave é suficiente para a classificação determinística de perguntas sem necessidade de um modelo de linguagem.

O *pipeline* RAG obteve 100% de exatidão no *golden set* de 50 perguntas, com tempo médio de 3,3 segundos.

O SQL Agent obteve 100% de taxa de execução em 100 perguntas distribuídas por três níveis de dificuldade (40 simples, 36 médias e 24 complexas), com tempo médio de 3,9 segundos. Ambos os valores de latência média ficam dentro dos requisitos não funcionais definidos (RNF 2: RAG < 5s e RNF 3: SQL < 15s).

5.5.4 Validação com Golden Set da Geodouro

A avaliação do SQL Agent foi realizada com recurso a um *golden set* fornecido pela equipa técnica da GeoDouro, composto por cinquenta perguntas representativas da utilização real do sistema, sendo trinta perguntas sobre a componente de monitorização e práticas agrícolas, e vinte perguntas sobre o módulo da adega. As perguntas cobrem desde listagens simples de parcelas e variedades até agregações mensais de horas trabalhadas, cálculos de estimativas e validações de operações de vinificação.

Para automatizar a avaliação das cinquenta respostas por iteração de otimização, foi adotado o método *LLM-as-Judge*, no qual um modelo de linguagem de maior capacidade funciona como avaliador automático. O modelo avaliador utilizado foi o Qwen2.5 14B. Para cada pergunta do *golden set*, o avaliador recebeu a pergunta original, a resposta do chatbot e a instrução SQL executada, atribuindo uma pontuação binária, 1 se a resposta fosse correta, e 0 caso contrário. Esta avaliação fez-se de acordo com quatro critérios. No primeiro, a resposta apresenta dados relevantes ou indica corretamente a ausência de registos. No segundo, não existe código SQL exposto ao utilizador. No terceiro, não existem mensagens de erro técnico visíveis, e por último, o SQL gerado é semanticamente compatível com a pergunta. A variabilidade estimada entre execuções independentes do mesmo conjunto é de uma a duas perguntas, justificada pelo não-determinismo residual do modelo avaliador. O sistema foi testado em cinco iterações de otimização sucessivas, permitindo quantificar o impacto isolado de cada conjunto de melhorias. A

Figura 18 ilustra a progressão ao longo das iterações.

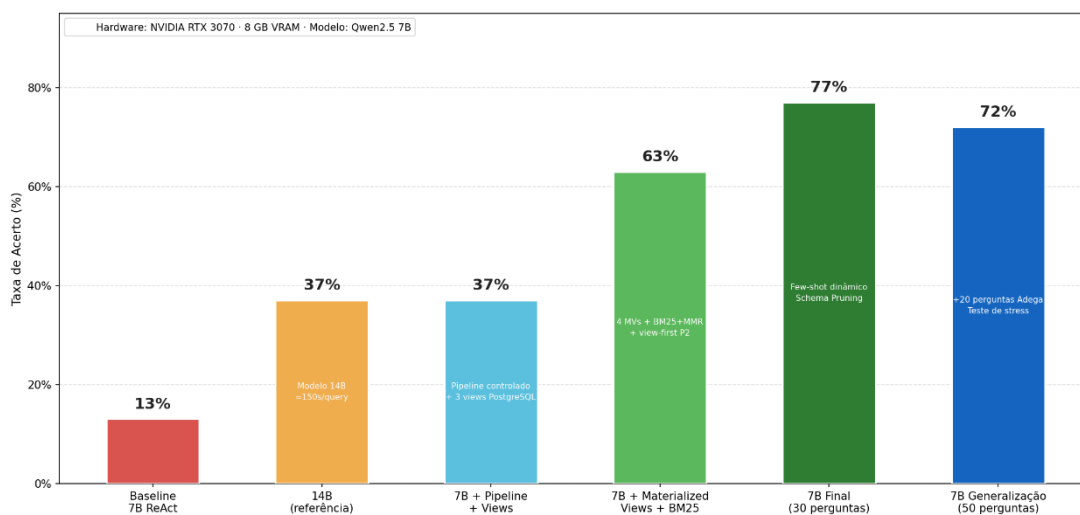


Figura 18 - Gráfico de evolução da taxa de acerto no golden set

A Tabela 6 resume os resultados obtidos.

Tabela 6 - Evolução da taxa de acerto no golden set por iteração de otimização

Iteração	Principais Alterações	Taxa de acerto
<i>Baseline</i> 7B com ReAct	Agente ReAct, sem vistas	13%
Modelo 14B (referência)	Modelo 14B, 150 s por pergunta	37%
7B + Pipeline + Vistas	Pipeline + 3 vistas PostgreSQL	37%
7B + MVs + BM25+MMR	4 vistas materializadas + pesquisa híbrida	63%
7B Final (30 perguntas)	<i>Few-shot</i> dinâmico + <i>schema pruning</i>	77%
7B Generalização (50 perguntas)	+20 perguntas adega	72%

Os resultados demonstram que o tamanho do modelo não é o fator determinante para a qualidade das respostas, o modelo de 14B parâmetros obteve o mesmo resultado que a terceira iteração com 7B parâmetros, confirmando que o ganho real provém da arquitetura do pipeline e das otimizações de recuperação. A versão final com 7B supera o modelo de referência de 14B em 40% nas trinta perguntas de referência, com tempos de resposta dez a trinta vezes inferiores.

Para validar a capacidade de generalização, o *golden set* foi alargado com as vinte perguntas do módulo de adega. A taxa de acerto no conjunto completo de cinquenta perguntas foi de 72%. Dos 28% de perguntas com avaliação negativa, 8 resultam de filtros incorretos ou de roteamento para a vista errada, falhas corrigíveis com regras adicionais no *prompt*, e 5 correspondem a limitações estruturais da base de dados, nomeadamente a ausência de uma relação direta entre certas tabelas. Os 3 casos restantes correspondem a respostas do *fallback* ReAct que expuseram raciocínio interno ao utilizador, o que viola as regras de apresentação.

5.5.5 Avaliação Comparativa de Estratégias de Retrieval

Para fundamentar a escolha da estratégia de *retrieval*, foi feita uma avaliação de seis configurações distintas sobre o mesmo *golden set* de 50 perguntas. Com o intuito de contextualizar as abordagens testadas, apresenta-se de seguida uma breve descrição do fundamento técnico de cada metodologia:

- MMR Puro (*Maximal Marginal Relevance* - Base): Este algoritmo atua sobre embeddings e pondera a relevância semântica de cada documento face à pergunta contra a sua redundância em relação aos elementos já selecionados. O objetivo consiste em maximizar a diversidade da informação introduzida no contexto do modelo.
- HyDE (Hypothetical Document Embeddings): Esta técnica recorre ao modelo de linguagem para gerar uma resposta hipotética e potencialmente com imperfeições à questão do utilizador. De seguida, o sistema extrai o vetor dessa resposta idealizada para realizar a pesquisa no *vectorstore* (a base de dados vetorial), o que mitiga a distância geométrica habitual entre a semântica de uma pergunta e a de um documento afirmativo.
- Estratégias Híbridas (50/50, 30/70 e 70/30): Estas três configurações baseiam-se na fusão de duas formas de recuperação. A primeira é a pesquisa lexical clássica através do algoritmo BM25 sensível à correspondência exata de palavras-chave e terminologia especializada, e a segunda é a pesquisa semântica densa via MMR. Os valores numéricos representam, respetivamente, a distribuição percentual de peso atribuída a cada motor na fórmula final de ordenação de resultados.
- Reranker (Reordenação): Esta abordagem divide-se em duas etapas lógicas. Na primeira fase, um recuperador vetorial extrai um volume largo de documentos candidatos, com vista a garantir uma alta taxa de recuperação. Na segunda fase, um modelo de classificação mais robusto e computacionalmente pesado avalia a atenção detalhada entre a pergunta e cada candidato, e reordena a listagem para reter apenas os três fragmentos com maior relevância absoluta.

Os resultados numéricos encontram-se na Tabela 7 e a análise visual nas Figura 19 a Figura 21.

Tabela 7 - Comparação de estratégias de retrieval

Estratégia	Precisão	Sem resposta	Tempo médio
Baseline (MMR)	86,0%	14,0%	2,7s
HyDE	80,0%	20,0%	8,6s
Hybrid 50/50	92,0%	8,0%	2,5s
Hybrid 30/70	90,0%	10,0%	2,6s
Hybrid 70/30	94,0%	6,0%	2,6s
Rerank	90,0%	10,0%	5,8s

A Tabela 7 mostra que o hybrid 70/30 obteve a maior taxa de acerto com 94% com uma latência média de 2,6 segundos. O HyDE foi a estratégia com pior desempenho, tanto em acerto (80%) como em latência (8,6s). O reranker melhorou o acerto face ao *baseline* em 4%, mas com latência 2,1 vezes superior. A análise detalhada por estratégia encontra-se nas figuras seguintes.

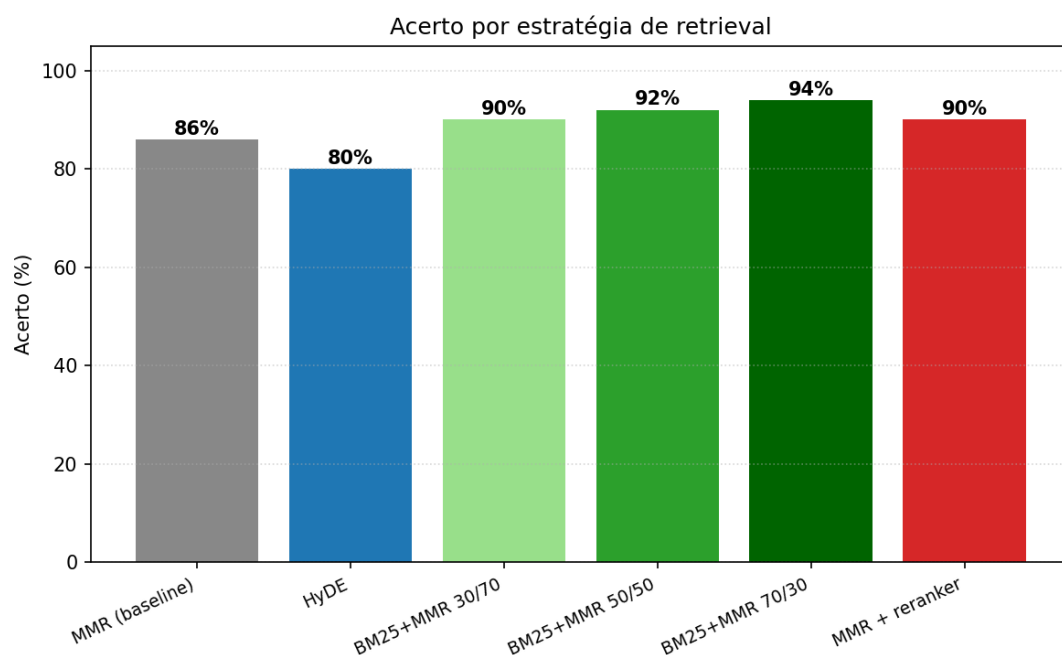


Figura 19 - Acerto por estratégia de *retrieval*

A Figura 19 apresenta a taxa de acerto de cada estratégia de *retrieval* no *golden set* de 50 perguntas. A estratégia *hybrid 70/30* obteve o melhor resultado com 94%, seguida pelo *hybrid 50/50* com 92%. O *baseline MMR* ficou em 86%, o *reranker* e o *hybrid 30/70* em 90%, e o *HyDE* obteve o pior resultado com 80%. A diferença de 14% entre a melhor e a pior estratégia demonstra que a escolha do *retriever* tem impacto direto na qualidade das respostas, independentemente do modelo de linguagem utilizado.

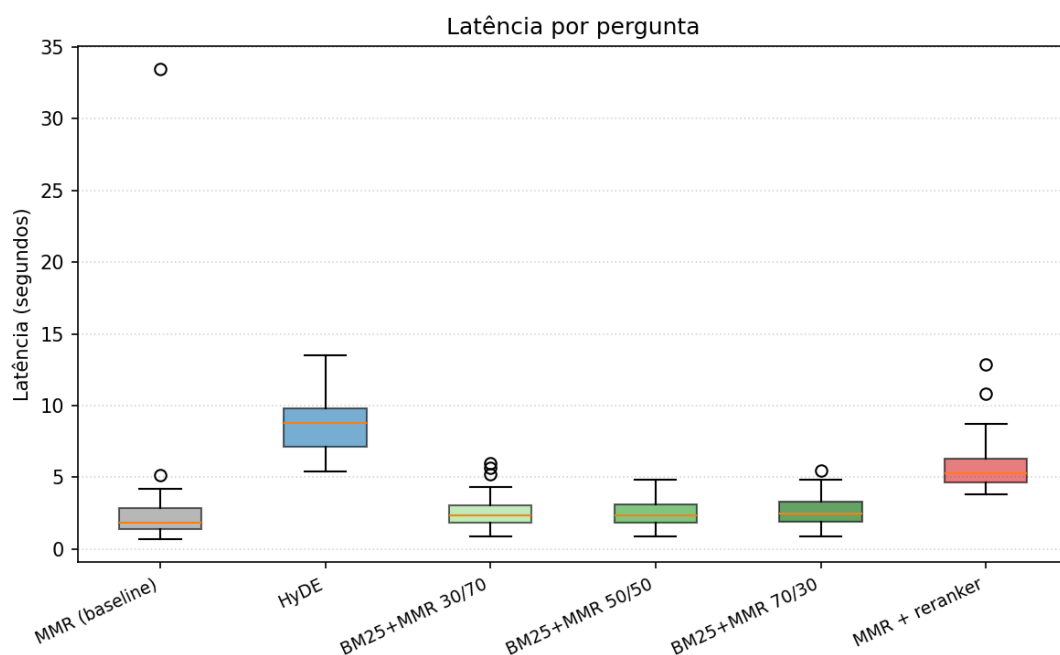


Figura 20 – Latência por pergunta

A Figura 20 compara a latência média e máxima de cada estratégia. As estratégias híbridas e o *baseline* apresentam latências médias semelhantes, entre 2,5 e 2,7 segundos, o que indica que a adição do componente BM25 não penaliza o tempo de resposta face ao MMR puro. O *reranker* apresenta uma latência média de 5,8 segundos, mais do dobro das estratégias híbridas, devido ao passo adicional de reordenação dos resultados. O HyDE é a estratégia mais lenta com 8,6 segundos de média, resultado da geração de um documento hipotético antes de cada pesquisa. Os valores máximos refletem o arranque a frio do modelo em todas as estratégias.

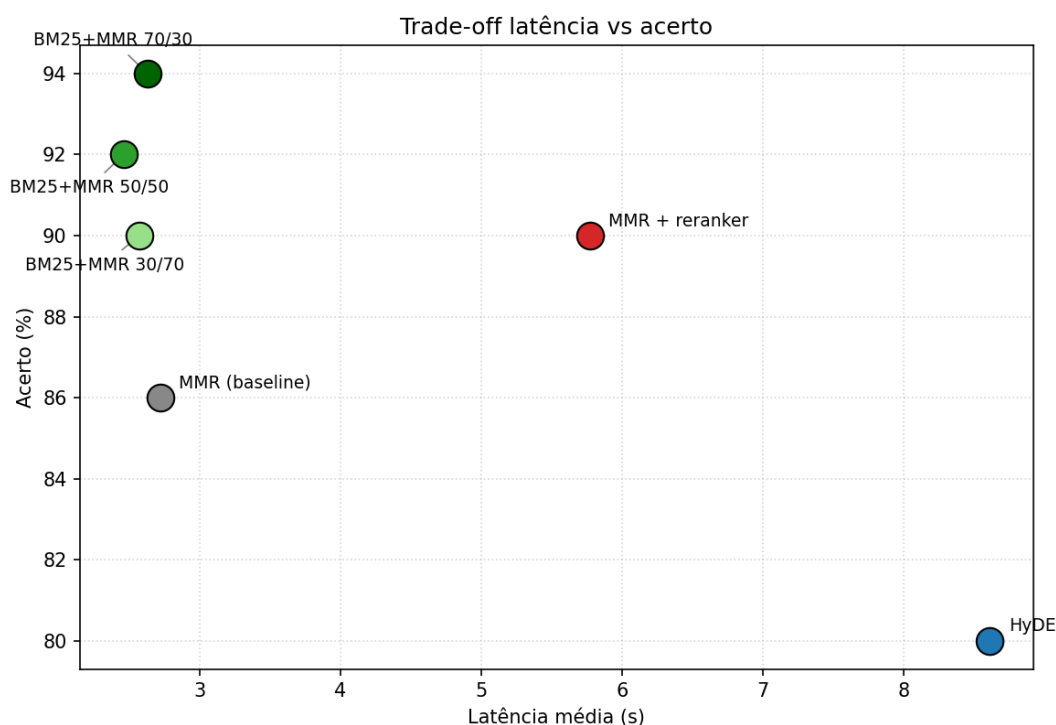


Figura 21 - Latência vs Acerto

A Figura 21 representa cada estratégia num plano de acerto *versus* latência, onde o canto superior esquerdo corresponde à zona ideal (alta taxa de acerto e baixa latência). As três estratégias híbridas formam um conjunto próximo desse canto, com taxas de acerto entre 90% e 94% e latências entre 2,5 e 2,7 segundos. O reranker e o HyDE afastam-se para a direita sem ganho proporcional em acerto, o que os exclui como opções viáveis. O hybrid 70/30 é o ponto dominante do gráfico, sem nenhuma outra estratégia com melhor combinação de acerto e latência em simultâneo, o que justifica a sua adoção como configuração definitiva do sistema.

5.5.6 Avaliação do Vectorstore de Tabelas

A eficácia do enriquecimento semântico das tabelas foi validada com um teste comparativo entre dois *vectorstores*, o baseline (apenas nome da tabela e lista de colunas) e o enriquecido (baseline com descrições geradas pelo modelo Qwen2.5 7B a partir do *schema* e de três linhas de amostra de cada tabela). O teste usou 25 perguntas de referência distribuídas pelas três bases de dados, cada uma com a lista de tabelas esperadas nos primeiros cinco resultados.

O *vectorstore* base recuperou corretamente 63,0% das tabelas esperadas e o enriquecido recuperou 73,9%, observando-se uma melhoria de 10,9%. Não foram identificadas regressões reais. Com base nestes resultados, o enriquecimento foi adotado como versão definitiva do índice de tabelas.

5.5.7 Testes de Escalabilidade

Foram realizados dois testes de escalabilidade distintos: um teste de carga sobre o *endpoint* de chat e um teste de dimensionamento do *vectorstore*.

O teste de dimensionamento do *vectorstore* avaliou o impacto do crescimento do índice RAG na latência de pesquisa e no tempo de indexação, testando sete volumes de *chunks* entre 200 e 12.000 conforme ilustrado na Figura 22. A latência média de pesquisa manteve-se estável entre 27 e 29 milissegundos até aos 6.000 *chunks*, com um ligeiro aumento para 39 milissegundos aos 12.000 *chunks*. O percentil 95 permaneceu entre 30 e 36 milissegundos em todos os volumes, o que confirma que o modelo de *embeddings* (intfloat/multilingual-e5-small) executado em CPU não constitui um estrangulamento de pesquisa na escala prevista para o sistema. O tempo de indexação cresce de forma linear: 11 segundos para 200 *chunks*, 51 segundos para 1.000 *chunks* e 1.062 segundos para 12.000 *chunks*. O índice atual do projeto contém cerca de 1.557 *chunks* reais, o que resulta num tempo de indexação de 85 segundos durante a inicialização do servidor.

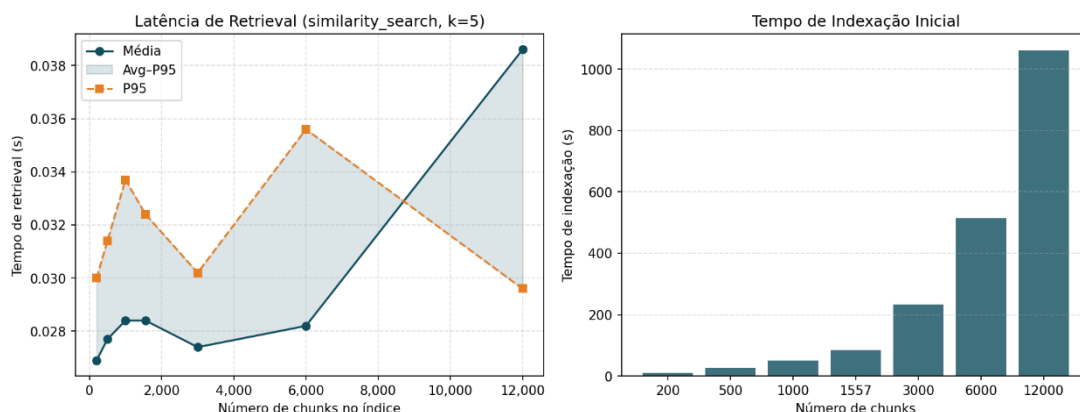


Figura 22 - Escalabilidade do vectorstore ChromaDB: impacto do número de chunks na latência de pesquisa (esquerda) e no tempo de indexação inicial (direita)

Adicionalmente, foram realizados testes de carga para avaliar o comportamento do sistema com múltiplas sessões em simultâneo. O servidor Ollama serializa os pedidos de inferência na GPU disponível (NVIDIA RTX 3070 com 8 GB de VRAM), pelo que o paralelismo de inferência não é suportado nativamente ao nível do hardware e, consequentemente, os pedidos em fila aguardam a conclusão do pedido anterior. O impacto traduz-se num aumento aproximadamente linear da latência face ao número de utilizadores em simultâneo, embora sem qualquer perda de estabilidade ou registo de falhas (taxa de erro de 0% no servidor backend).

Os testes simularam um cenário realista ao combinar a submissão simultânea de perguntas mistas de RAG (cerca de 75%) e consultas SQL (cerca de 25%), procedendo-se à medição da latência de cada resposta. Num cenário de base com 1 utilizador, o tempo mediano de resposta situou-se nos 5,2 segundos, com uma latência máxima registada de 19,5 segundos. Ao aumentar a carga para 5 utilizadores em simultâneo, o tempo mediano de resposta subiu para os 24,0 segundos, observando-se uma latência máxima de 40,4 segundos (ver Figura 23).

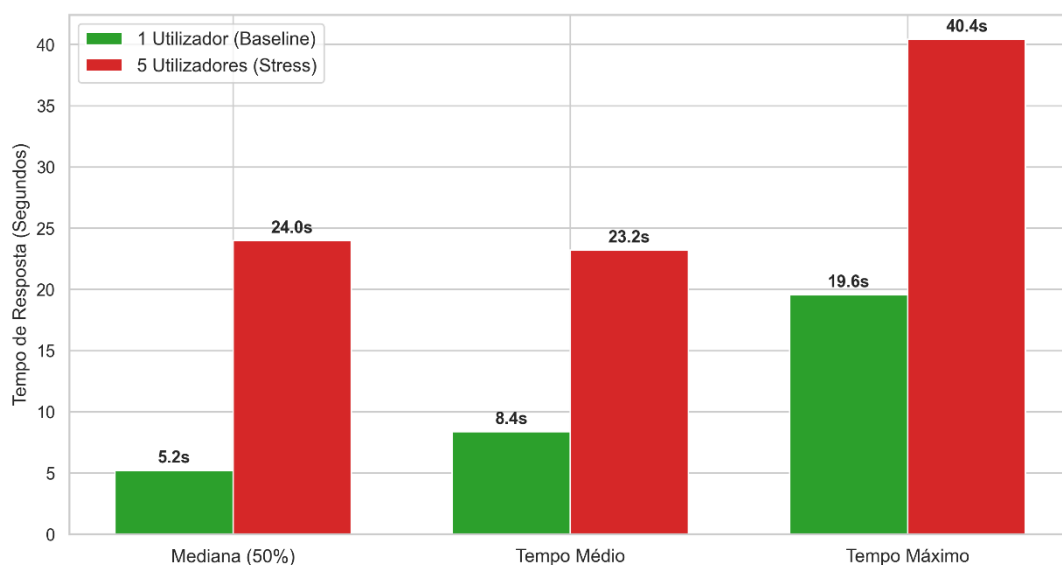


Figura 23 - Comparação dos tempos de resposta do sistema entre um cenário de baseline (1 utilizador) e um cenário de stress (5 utilizadores concorrentes).

Considerando o âmbito de utilização previsto, um número reduzido de utilizadores por entidade numa plataforma interna de gestão agrícola, conclui-se que este comportamento é perfeitamente adequado. Contudo, como ilustrado na Figura 24, a carga sustentada de 5 utilizadores eleva o teto máximo de resposta ao longo do tempo.

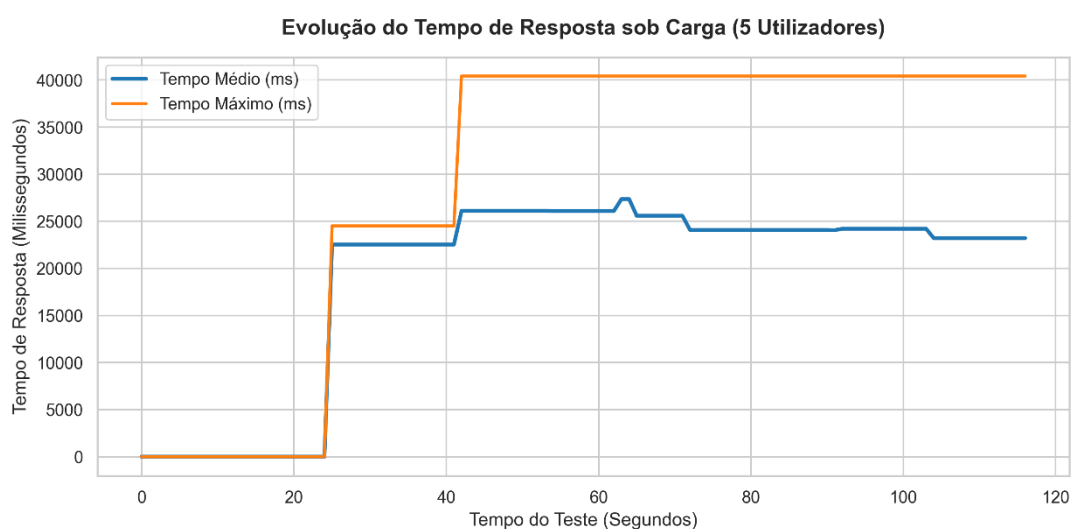


Figura 24 - Evolução da latência cumulativa ao longo da execução do teste de carga com 5 utilizadores concorrentes

5.6 Fase 6: Finalização

A sexta e última fase foi dedicada à documentação técnica do sistema. A elaboração da documentação técnica assumiu um papel central na fase final do estágio. O objetivo consistiu em garantir a transição fluida do projeto para a GeoDouro, assegurando a sua operação e manutenção futuras sem dependência do autor original. Para tal, foi redigido um documento de software exaustivo, um ficheiro README.md, integrado diretamente no repositório do código.

O documento descreve a organização estrutural do projeto, que se encontra dividido em módulos lógicos. Destacam-se as pastas de configuração e vetorização, os fluxos de execução de inteligência artificial, a interface de comunicação de servidor e a interface visual do utilizador.

Além da arquitetura, o documento especifica os requisitos de sistema, como a necessidade de uma unidade de processamento gráfico com um mínimo de 8 GB de VRAM, e detalha os passos precisos para a instalação e configuração de variáveis de ambiente. Um elemento crítico incluído nesta documentação é o procedimento de preparação das bases de dados operacionais. O documento fornece as *queries* SQL exatas para instanciar as vistas simples e as vistas materializadas indispensáveis ao agente SQL, e instrui sobre os comandos necessários para a sua atualização após a ingestão de novos dados no SIGP.

Por fim, a documentação incorpora um guia de operação do painel de administração e uma secção avançada de resolução de problemas. Esta componente capacita a equipa da empresa para adicionar novas bases de dados à arquitetura, forçar a reindexação vetorial do conhecimento e diagnosticar falhas comuns, como tempos de resposta elevados ou respostas sem informação suficiente.

6 Conclusão

O presente capítulo apresenta as conclusões do estágio, sintetizando os objetivos alcançados, os resultados obtidos e as dificuldades encontradas ao longo do desenvolvimento. São ainda descritas as aprendizagens adquiridas e as propostas de trabalho futuro.

6.1 Objetivos Alcançados

Os objetivos definidos no início do estágio foram atingidos na sua totalidade. O sistema entregue é um *chatbot* inteligente integrado ao SIGP, capaz de responder a perguntas sobre as funcionalidades da plataforma e de consultar dados reais das bases de dados em linguagem natural. Os três objetivos principais foram concretizados: a construção de um sistema de pergunta e resposta sobre documentação (RAG), a construção de um agente SQL capaz de gerar e executar *queries* a partir de linguagem natural, e a integração de ambos numa interface web de utilização simples.

O sistema foi implementado com processamento 100% local, sem envio de dados para serviços externos, em cumprimento do requisito de privacidade estabelecido pela empresa. A interface web oferece *streaming* de respostas, histórico de sessões persistente e um painel de administração que permite atualizar a documentação sem intervenção técnica. A observabilidade foi assegurada através do Langfuse em modo *self-hosted*, com registo de todos os *traces* de execução e um mecanismo de *feedback* de utilizador integrado na interface.

6.2 Resultados e Contribuições

Os resultados obtidos confirmam a eficácia da arquitetura desenvolvida e o cumprimento integral dos requisitos não funcionais. No cenário de consulta a documentação (pipeline RAG), o sistema alcançou uma taxa de acerto de 100% no conjunto de testes de referência, com um tempo médio de resposta de 3,3 segundos. No cenário de consulta a bases de dados (SQL Agent), o tempo médio fixou-se nos 3,9 segundos. Ambos os valores cumprem com grande margem os limites estabelecidos de 5 e 15 segundos, respetivamente.

Estes indicadores validam de forma clara as decisões arquiteturais tomadas ao longo do projeto. Em primeiro lugar, o router de classificação, implementado com base em regras estritas em vez de modelos de linguagem, acertou todas as perguntas dos testes unitários com latência quase nula. Esta escolha comprova que tarefas determinísticas beneficiam de abordagens algorítmicas clássicas, produzindo resultados mais robustos do que a IA generativa.

Em segundo lugar, a avaliação comparativa de recuperação de informação revelou que a estratégia híbrida BM25+MMR (com peso 70/30) é a mais adequada para lidar com documentação técnica em português de Portugal, atingindo 94% de precisão sem penalizar a latência. No domínio das bases de dados, a inovação de enriquecer semanticamente o esquema com descrições geradas localmente traduziu-se num aumento direto de 10,9% na recuperação de tabelas corretas por parte do agente SQL.

Por fim, a aposta numa plataforma de observabilidade revelou-se um fator crítico para a viabilidade do produto. A análise detalhada dos registos no Langfuse permitiu diagnosticar e corrigir o problema do recarregamento constante do modelo Ollama na placa gráfica. Esta intervenção reduziu drasticamente a latência de picos de 67 segundos para um intervalo estável de 2 a 4 segundos. Sem este nível de instrumentação, falhas operacionais ocultas desta natureza seriam impossíveis de identificar em ambiente de produção.

6.3 Dificuldades Encontradas

A principal dificuldade foi a limitação de memória de vídeo, apenas 8 GB VRAM condicionou várias decisões técnicas. O problema mais impactante foi o recarregamento automático do modelo Ollama quando o parâmetro `num_ctx` diferia entre componentes, com pausas de 30 segundos em cada transição. A solução só foi identificada após a análise dos *traces* no Langfuse, o que sublinhou a importância da observabilidade desde o início do desenvolvimento.

A escala das bases de dados tornou inviável a escrita manual de descrições semânticas. A solução foi automatizar a geração com o próprio modelo Qwen2.5 7B, mas o processo exigiu vários ciclos de validação para garantir que as descrições eram suficientemente informativas sem introduzir informação incorreta.

A gestão das alucinações do SQL Agent foi uma dificuldade persistente. O agente tendia a reutilizar dados do histórico de sessão em vez de executar novas *queries*, e a gerar nomes de colunas inexistentes quando não consultava o *schema*. Cada problema exigiu uma intervenção específica no *system prompt* ou na lógica do agente, com validação nos *traces* antes de ser considerado resolvido.

6.4 Aprendizagens Adquiridas

O estágio proporcionou contacto prático com o ciclo completo de desenvolvimento de um sistema baseado em modelos de linguagem desde a ingestão de documentos e a construção de *vectorstores* até ao *deployment* de uma API com *streaming* e à avaliação quantitativa da qualidade das respostas.

Do ponto de vista técnico, as aprendizagens mais relevantes foram o desenho de *pipelines* RAG com LangChain, a implementação de agentes ReAct com LangGraph, o *prompt engineering* para SQL e RAG, e a instrumentação com Langfuse. A experiência mostrou que o *prompt engineering* é uma disciplina iterativa, cada regra adicionada ao *system prompt* resulta de um problema concreto observado em produção, não de antecipação teórica.

Uma aprendizagem transversal foi a importância de medir antes de otimizar. As duas otimizações com maior impacto no projeto, a unificação do `num_ctx` e enriquecimento semântico das tabelas, foram identificadas por dados, não por intuição. Sem o sistema de observabilidade, o problema do recarregamento do modelo provavelmente teria sido atribuído a causas erradas e resolvido de forma ineficaz.

O trabalho com bases de dados reais de uma aplicação agrícola em produção trouxe também uma perspetiva sobre a complexidade do domínio, com cerca de 500 tabelas com nomes técnicos, relações entre bases de dados distintas e dados com formatos heterogéneos, sendo assim um contexto muito diferente dos exemplos académicos, e exigindo soluções que os recursos disponíveis não cobriam diretamente.

6.5 Trabalho Futuro

O desenvolvimento deste projeto estabeleceu uma arquitetura robusta e funcional. Contudo, a sua passagem para um ambiente de produção em larga escala abre portas a diversas oportunidades de evolução técnica e funcional. As propostas de trabalho futuro estruturam-se em quatro eixos principais:

- **Integração e Experiência de Utilizador:** A aplicação atual opera de forma independente. O passo seguinte para melhorar a adoção do sistema passa pela sua integração nativa no portal web e na aplicação do SIGP, através de *Web Components* ou *iframes*. Em paralelo, torna-se imperativa a implementação de um sistema de autenticação ligado ao controlo de acessos do SIGP, o que assegura que cada utilizador apenas pergunta os dados da sua própria entidade. Adicionalmente, a exploração de um suporte multimodal com reconhecimento de voz através do modelo Whisper e síntese de fala via Coqui TTS proporcionará uma interação natural em contexto de campo, onde a utilização de teclados físicos é limitativa.

- **Escalabilidade de Hardware e Fine-Tuning:** Os testes de carga demonstraram que a unidade de processamento gráfico utilizada (RTX 3070 com 8 GB VRAM) processa os pedidos de forma sequencial, o que aumenta a latência em cenários com mais de três utilizadores simultâneos. A migração do sistema para uma infraestrutura de servidor com GPU de maior capacidade resolverá este constrangimento através de processamento paralelo, e viabilizará a execução de modelos mais robustos de 14B ou 32B parâmetros. Como alternativa a modelos maiores, equaciona-se o fine-tuning supervisionado do próprio modelo Qwen2.5 7B com pares de perguntas e consultas SQL específicos do domínio vitivinícola, com vista a erradicar a confusão entre colunas de esquemas semelhantes sem agravar os requisitos de hardware.
- **Otimização do Modelo de Dados:** No plano da base de dados, a manutenção do sistema beneficiará da automatização do processo de atualização das vistas materializadas. A configuração de tarefas calendarizadas nativas no PostgreSQL eliminará a dependência atual de atualizações manuais. Ademais, a resolução dos dois casos de uso não suportados no módulo de adega requererá uma intervenção direta no modelo relacional do próprio SIGP para estabelecer relações transacionais explícitas.
- **Avaliação Contínua:** A melhoria sistemática do modelo exige a expansão do *golden set* atual com perguntas e interações reais recolhidas em produção. A incorporação desta base de dados alargada num ciclo de avaliação automática, com recurso à técnica LLM-as-a-Judge, possibilitará a deteção imediata de regressões na qualidade das respostas após qualquer atualização de código ou alteração documental.

Referências Bibliográficas

- Ahia, O., Kumar, S., Gonen, H., Kasai, J., Mortensen, D. R., Smith, N. A., & Tsvetkov, Y. (2023). Do All Languages Cost the Same? Tokenization in the Era of Commercial Language Models. *arXiv:2305.13707*.
- Cho, K., Merrienboer, B. van, Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). *Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation*. arXiv. <https://doi.org/10.48550/arXiv.1406.1078>
- Chroma. (2026). Chroma. <https://www.trychroma.com>
- Comparison of FastAPI with Django and Flask. (2025). GeeksforGeeks. <https://www.geeksforgeeks.org/python/comparison-of-fastapi-with-django-and-flask/>
- Cormack, G. V., Clarke, C. L. A., & Buettcher, S. (2009). Reciprocal rank fusion outperforms condorcet and individual rank learning methods. *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval, SIGIR '09*, 758–759. <https://doi.org/10.1145/1571941.1572114>
- DeepSeek-AI. (2024). DeepSeek-V2: A Strong, Economical, and Efficient Mixture-of-Experts Language Model. *arXiv:2405.04434*.
- Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv:1810.04805*.
- Didwania, K., Seth, P., Kasliwal, A., & Agarwal, A. (2024). *AgriLLM: Harnessing Transformers for Farmer Queries* (arXiv:2407.04721; Versão 1). arXiv. <https://doi.org/10.48550/arXiv.2407.04721>

- Docker*. (2025, abril 9). <https://www.docker.com/>
- Fanuel, M., Mahmoud, M. N., Marshal, C. C., Lakhota, V., Dari, B., Roy, K., & Zhang, S. (2025). *AgriRegion: Region-Aware Retrieval for High-Fidelity Agricultural Advice* (arXiv:2512.10114; Versão 1). arXiv. <https://doi.org/10.48550/arXiv.2512.10114>
- FastAPI*. (2026). Obtido 28 de março de 2026, de <https://fastapi.tiangolo.com/>
- Frantar, E., Ashkboos, S., Hoefler, T., & Alistarh, D. (2023). *GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers* (arXiv:2210.17323). arXiv. <https://doi.org/10.48550/arXiv.2210.17323>
- Gao, Y., Xiong, Y., Gao, X., Jia, K., Pan, J., Bi, Y., Dai, Y., Sun, J., Wang, M., & Wang, H. (2024, março). *Retrieval-Augmented Generation for Large Language Models: A Survey*. arXiv. <https://doi.org/10.48550/arXiv.2312.10997>
- Git*. (2026). <https://git-scm.com/>
- GitHub*. (2026). GitHub. <https://github.com>
- GPP. (2026). *PEPAC | Plano Estratégico da PAC 2023-2027*. <https://www.gpp.pt/index.php/pepac/pepac-plano-estrategico-da-pac-2023-2027>
- Gupta, S., Ranjan, R., & Singh, S. N. (2024). *A Comprehensive Survey of Retrieval-Augmented Generation (RAG): Evolution, Current Landscape and Future Directions* (arXiv:2410.12837). arXiv. <https://doi.org/10.48550/arXiv.2410.12837>
- Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8), 1735–1780. <https://doi.org/10.1162/neco.1997.9.8.1735>
- GeoDouro*. (2026). <https://geodouro.pt/>
- Huang, D., & Wang, Z. (2025). LLMs at the Edge: Performance and Efficiency Evaluation with Ollama on Diverse Hardware. *2025 International Joint Conference on Neural Networks (IJCNN)*, 1–8. <https://doi.org/10.1109/IJCNN64981.2025.11228317>
- Ji, Z., Lee, N., Frieske, R., Yu, T., Su, D., Xu, Y., Ishii, E., Bang, Y. J., Madotto, A., & Fung, P. (2023). Survey of Hallucination in Natural Language Generation. *ACM Computing Surveys*, 55(12).

- Jiang, A. Q., Sablayrolles, A., Mensch, A., Bamford, C., Chaplot, D. S., de las Casas, D., Bressand, F., Lengyel, G., Lample, G., Saulnier, L., Lavaud, L. R., Lachaux, M. A., Stock, P., Le Scao, T., Lavril, T., Wang, T., Lacroix, T., & El Sayed, W. (2023). Mistral 7B. *arXiv:2310.06825*.
- Jurafsky, D., & Martin, J. H. (2023). *Speech and Language Processing* (3rd ed.). <https://web.stanford.edu/~jurafsky/slp3/>
- LangChain*. (2026). Docs by LangChain. <https://docs.langchain.com/oss/python/langchain/overview>
- Langfuse*. (2026). <https://langfuse.com/?tab=metrics>
- LangGraph*. (2026). Docs by LangChain. <https://docs.langchain.com/oss/python/langgraph/overview>
- Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W. T., Rocktäschel, T., Riedel, S., & Kiela, D. (2020). Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. *arXiv:2005.11401*.
- Liu, N. F., Lin, K., Hewitt, J., Paranjape, A., Bevilacqua, M., Petroni, F., & Liang, P. (2023). Lost in the Middle: How Language Models Use Long Contexts. *arXiv:2307.03172*.
- Ma, L., Zhang, R., Han, Y., Yu, S., Wang, Z., Ning, Z., Zhang, J., Xu, P., Li, P., Ju, W., Chen, C., Wang, D., Liu, K., Wang, P., Wang, P., Fu, Y., Liu, C., Zhou, Y., & Lu, C.-T. (2025). *A Comprehensive Survey on Vector Database: Storage and Retrieval Technique, Challenge* (arXiv:2310.11703; Versão 2). arXiv. <https://doi.org/10.48550/arXiv.2310.11703>
- Manifesto for Agile Software Development*. (2026). <https://agilemanifesto.org/>
- Meta AI. (2024a). *Llama 3 Model Card*. <https://huggingface.co/meta-llama/Meta-Llama-3.1-8B-Instruct>
- Meta AI. (2024b). *Llama 3.1 405B Model Card*. <https://huggingface.co/meta-llama/Llama-3.1-405B>

- Mittal, A., Kumar, M., & Garg, A. (2025). *Artificial intelligence in agriculture: A review of applications, challenges and future directions*.
- NetworkX. (2026). <https://networkx.org/en/>
- Nie, Z., Feng, Z., Li, M., Zhang, C., Zhang, Y., Long, D., & Zhang, R. (2025). *When Text Embedding Meets Large Language Model: A Comprehensive Survey* (arXiv:2412.09165; Versão 3). arXiv. <https://doi.org/10.48550/arXiv.2412.09165>
- OpenAI. (2026). *What are tokens and how to count them?* <https://help.openai.com/en/articles/4936856-what-are-tokens-and-how-to-count-them>
- OpenAPI Specification. (2021). <https://spec.openapis.org/oas/v3.1.0>
- Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., Schulman, J., Hilton, J., Kelton, F., Miller, L., Simens, M., Askell, A., Welinder, P., Christiano, P., Leike, J., & Lowe, R. (2022). Training language models to follow instructions with human feedback. *arXiv:2203.02155*.
- Paul, A., & Machavaram, R. (2025). Generative AI in agriculture 4.0: Applications, challenges, and integration in the Indian context. *Food and Humanity*, 5, 100889. <https://doi.org/https://doi.org/10.1016/j.foohum.2025.100889>
- Postman. (2026). <https://www.postman.com/>
- Psycopg. (2026). <https://www.psycopg.org/docs/>
- Pydantic Validation. (2026). <https://docs.pydantic.dev/latest/>
- Qwen Team. (2025). Qwen3 Technical Report. *arXiv:2505.09388*.
- Qwen, Yang, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Li, C., Liu, D., Huang, F., Wei, H., Lin, H., Yang, J., Tu, J., Zhang, J., Yang, J., Yang, J., Zhou, J., Lin, J., ... Qiu, Z. (2025). *Qwen2.5 Technical Report* (arXiv:2412.15115). arXiv. <https://doi.org/10.48550/arXiv.2412.15115>

- Raschka, S., Patterson, J., & Nolet, C. (2020). *Machine Learning in Python: Main developments and technology trends in data science, machine learning, and artificial intelligence* (arXiv:2002.04803). arXiv. <https://doi.org/10.48550/arXiv.2002.04803>
- Sapkota, R., Qureshi, R., Usman Hadi, M., Zohaib Hassan, S., Sadak, F., Shoman, M., Sajjad, M., Ali Dharejo, F., Paudel, A., Li, J., Meng, Z., Shutske, J., & Karkee, M. (2025). Multi-Modal LLMs in Agriculture: A Comprehensive Review. *IEEE Transactions on Automation Science and Engineering*, 22, 22510–22540. <https://doi.org/10.1109/TASE.2025.3612154>
- Scrum Guides. (2026). <https://scrumguides.org/>
- Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q., Hinton, G., & Dean, J. (2017). Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer. *arXiv:1701.06538*.
- SIFITO. (2026). Obtido 16 de março de 2026, de <https://sifito.dgav.pt/Account/Login?ReturnUrl=%2F>
- SIGP. (2026). Obtido 16 de março de 2026, de <https://www.sigp.pt/>
- Stability AI. (2023). *StableLM Zephyr 3B*. <https://huggingface.co/stabilityai/stablelm-zephyr-3b>
- The Python Standard Library. (2026). Python Documentation. <https://docs.python.org/3/library/asyncio.html>
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). *Attention is all you need* (arXiv:1706.03762). arXiv. <https://doi.org/10.48550/arXiv.1706.03762>
- Visual Studio Code. (2026). <https://code.visualstudio.com/>
- Wang, L., Yang, N., Huang, X., Yang, L., Majumder, R., & Wei, F. (2024). *Multilingual E5 Text Embeddings: A Technical Report*. <https://arxiv.org/abs/2402.05672v1>
- Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., Platen, P. von, Ma, C., Jernite, Y., Plu, J.,

- Xu, C., Scao, T. L., Gugger, S., ... Rush, A. M. (2020). *HuggingFace's Transformers: State-of-the-art Natural Language Processing* (arXiv:1910.03771). arXiv. <https://doi.org/10.48550/arXiv.1910.03771>
- Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., & Cao, Y. (2022, outubro 6). *ReAct: Synergizing Reasoning and Acting in Language Models*. arXiv.Org. <https://arxiv.org/abs/2210.03629v3>

Anexos

Comparativo das 7 LLM

Alexandre Rodrigues Fernandes

Março 2026

1 Processamento de Linguagem Natural

O Processamento de Linguagem Natural (*Natural Language Processing*, PLN) é o ramo da inteligência artificial que estuda a interação entre computadores e a linguagem humana, com o objetivo de capacitar as máquinas a compreender, interpretar e gerar texto de forma significativa [2]. As suas aplicações abrangem desde a tradução automática e a análise de sentimentos até aos sistemas de perguntas e respostas que constituem o fundamento do chatbot desenvolvido neste projeto.

1.1 Redes Recorrentes e as suas Limitações

Durante décadas, a abordagem dominante em PLN baseou-se em redes neuronais recorrentes (*Recurrent Neural Networks*, RNN) e nas suas variantes mais sofisticadas, as *Long Short-Term Memory* (LSTM) [3] e as *Gated Recurrent Units* (GRU) [4]. Para serem processados por estas redes, os textos são primeiramente divididos em unidades fundamentais de informação designadas por *tokens*, que podem corresponder a uma palavra inteira, a um fragmento de palavra ou a um único carácter [23]. Estas arquiteturas processam o texto de forma estritamente sequencial, *token a token*, mantendo um estado interno que propaga informação ao longo da sequência.

No entanto, as redes recorrentes apresentam duas limitações estruturais que condicionaram o seu progresso:

- Problema do gradiente desvanescente - em sequências longas, o sinal de aprendizagem propaga-se com dificuldade para os passos iniciais da sequência, fazendo com que o modelo perca a capacidade de relacionar palavras distantes entre si [5]. As LSTM atenuam parcialmente este problema, mas não o eliminam.
- Processamento sequencial - a natureza passo-a-passo das RNN impede a paralelização do treino. O tempo de treino escala linearmente com o comprimento das sequências, tornando inviável o treino em corpora de larga escala com o hardware disponível.

1.2 A Arquitetura Transformer

Em 2017, Vaswani et al. [1] propuseram uma arquitetura radicalmente diferente: o Transformer. A inovação central é o mecanismo de atenção (*self-attention*), que permite a cada *token* de uma sequência relacionar-se diretamente com todos os outros *tokens* em paralelo, independentemente da distância entre eles. Esta propriedade resolve simultaneamente as duas limitações das RNN:

- As dependências de longo alcance são capturadas de forma direta, sem degradação do sinal.
- O processamento paralelo de toda a sequência permite escalar o treino para conjuntos de dados e modelos de dimensão anteriormente inviável.

A arquitetura Transformer tornou-se o suporte sobre o qual todos os modelos de linguagem modernos são construídos, desde o BERT [6] até às famílias Llama e Qwen avaliadas neste projeto.

2 Large Language Models

Um Modelo de Linguagem de Grande Escala (*Large Language Model*, LLM) é um sistema de inteligência artificial baseado na arquitetura *Transformer* [1], treinado em quantidades enormes de texto com o objetivo de modelar a distribuição probabilística da linguagem natural. Em termos práticos, um LLM aprende a prever qual o *token* mais provável a seguir a uma sequência prévia, o que lhe permite gerar texto coerente, responder a perguntas, resumir documentos e seguir instruções em linguagem natural.

A arquitetura *Transformer*, introduzida por Vaswani et al. em 2017 [1], revolucionou o processamento de linguagem natural ao substituir as redes recorrentes por mecanismos de atenção (*self-attention*) que processam toda a sequência de entrada em paralelo. Esta propriedade permite escalar o treino para conjuntos de dados e modelos de dimensão anteriormente inviável, originando os modelos modernos com dezenas ou centenas de mil milhões de parâmetros.

Os LLM modernos são tipicamente treinados em duas fases: um pré-treino não supervisionado que incorpora texto de larga escala (livros, páginas web, código, artigos científicos), seguido de um ajuste fino supervisionado com dados de instruções e, frequentemente, de um processo de alinhamento por reforço com feedback humano (*Reinforcement Learning from Human Feedback*, RLHF) [8]. O resultado são modelos capazes de generalizar para tarefas não vistas durante o treino, incluindo resposta a perguntas em domínios especializados quando complementados com contexto externo como no RAG (*Retrieval-Augmented Generation*) [9] utilizado neste projeto.

2.1 O que significam os parâmetros B?

A dimensão de um LLM é habitualmente expressa pelo número de parâmetros do modelo, ou seja, os valores numéricos (pesos) aprendidos durante o treino que determinam o comportamento do modelo. Embora os modelos estado da arte possuam frequentemente entre 70 e 400 mil milhões de parâmetros (como é o caso do Llama 3.1 405B [25]), a execução estritamente local em *hardware* de consumo impõe restrições severas de memória.

Por este motivo, o presente projeto foca-se exclusivamente na classe de modelos otimizados para execução local, cujo limite prático para inferência numa placa gráfica de 8GB de VRAM (como a utilizada no desenvolvimento) se situa na casa dos 8 mil milhões. Estes valores são habitualmente representados pela letra B de *billion* (mil milhões em inglês), destacando-se as seguintes dimensões avaliadas:

- 3B - 3 mil milhões de parâmetros (ex: StableLM-Zephyr 3B [17])
- 7B - 7 mil milhões de parâmetros (ex: Mistral 7B [15], Qwen2.5 7B [13])
- 8B - 8 mil milhões de parâmetros (ex: Llama 3.1 8B [12], Qwen3-8B [14])

Cada parâmetro ocupa memória: um modelo em precisão total (*float32*) requer 4 bytes por parâmetro, pelo que um modelo de 7B ocupa aproximadamente 28 GB. Para tornar estes modelos executáveis em *hardware* de consumo, recorre-se à quantização, uma técnica que reduz a precisão numérica dos pesos de 32 bits para 4 ou 8 bits, com perda mínima de qualidade [10]. Na quantização Q4_K_M utilizada neste projeto, cada parâmetro ocupa aproximadamente 0.5 bytes, reduzindo um modelo de 7B para cerca de 4–5 GB de VRAM.

A relação entre o número de parâmetros e a qualidade não é linear nem universal. Investigação recente demonstrou que modelos mais pequenos, treinados em dados de maior qualidade e durante mais tempo, superam frequentemente modelos maiores treinados de forma menos eficiente, fenómeno explorado pela família Mistral [15] e confirmado pelos resultados obtidos nos testes realizados neste projeto.

No caso dos modelos de arquitetura *Mixture-of-Experts* (MoE) [11], como o DeepSeek-V2-Lite [16], o número total de parâmetros é enganador: o modelo possui 15.7B parâmetros totais, mas apenas 2.4B são ativados por inferência, pois cada token é encaminhado apenas para um subconjunto de especialistas. Esta distinção é relevante para a estimativa de VRAM necessária.

2.2 Limitações dos LLM e a Necessidade de Conhecimento Externo

Apesar das capacidades notáveis dos LLM, a sua utilização em sistemas de suporte a domínios específicos, como o chatbot do SIGP, expõe três limitações fundamentais que inviabilizam a sua aplicação direta sem arquitetura complementar:

- Conhecimento estático - os LLM são treinados com dados até uma data de corte (*knowledge cutoff*). Qualquer informação posterior a essa data, incluindo atualizações ao software, novos procedimentos ou alterações funcionais, é desconhecida para o modelo [21].
- Ausência de conhecimento de domínio privado - a documentação interna de um sistema como o SIGP nunca fez parte dos dados de treino de nenhum modelo público. O modelo não tem qualquer conhecimento sobre os menus, procedimentos ou terminologia específica da aplicação.
- Alucinação - quando confrontado com uma pergunta para a qual não possui informação suficiente, um LLM tende a gerar respostas plausíveis mas factualmente incorretas, fenómeno designado por alucinação [7]. Num sistema de suporte agrícola, este comportamento é particularmente problemático: respostas incorretas sobre procedimentos como gestão de pragas podem induzir decisões com consequências reais.

É para mitigar estas três limitações que surge o paradigma RAG (*Retrieval-Augmented Generation*), descrito na secção seguinte.

2.3 Janela de Contexto e RAG

A janela de contexto (*context window*) de um LLM define o número máximo de tokens que o modelo consegue processar numa única inferência, tanto o texto de entrada (*prompt*) como a resposta gerada. De acordo com a OpenAI, um token corresponde tipicamente a 0,75 palavras em inglês [23]. Contudo, devido à mecânica dos algoritmos de tokenização (*Byte-Pair Encoding*), línguas como o português sofrem uma ligeira penalização, exigindo frequentemente entre 1,5 a 2 tokens por palavra [24]. Neste contexto conservador, uma janela de 4096 tokens equivale a cerca de 2000 a 2500 palavras em português, e uma janela de 128k tokens a aproximadamente 96 000 palavras [19].

Esta limitação é particularmente relevante no paradigma RAG (*Retrieval-Augmented Generation*) [9], que constitui a arquitetura base do chatbot desenvolvido neste projeto. No RAG, o modelo não possui conhecimento interno sobre o sistema em questão, em vez disso, para cada pergunta do utilizador, o sistema recupera os documentos mais relevantes da base vetorial (ChromaDB [20]) e insere-os diretamente no *prompt* enviado ao LLM.

O *prompt* enviado ao modelo é composto por três elementos que competem pela janela de contexto disponível:

- Prompt de sistema - instruções de comportamento do chatbot (língua, tom, restrições), tipicamente 100–300 tokens.
- Chunks recuperados - excertos dos documentos mais relevantes para a pergunta, com $k = 3$ chunks de 400 tokens cada, totalizam cerca de 1200 tokens.
- Pergunta e histórico de conversa - a pergunta atual e, opcionalmente, mensagens anteriores da conversa variável.

Uma janela de contexto de 4096 tokens (como a do StableLM-Zephyr 3B) é insuficiente para este caso, o espaço disponível para a resposta após subtrair o *prompt* de sistema, os chunks e a pergunta é inferior a 1000 tokens, o que limita severamente a capacidade de resposta e inviabiliza a inclusão de histórico conversacional. Modelos com janelas de 32k ou 128k tokens eliminam esta restrição, permitindo recuperar mais chunks, incluir contexto conversacional e gerar respostas mais completas [21].

A relação entre janela de contexto, número de chunks recuperados e qualidade da resposta RAG foi estudada por Gao et al. [21], que demonstraram que aumentar k (número de chunks) melhora a qualidade da resposta até um ponto de saturação, a partir do qual a informação redundante ou irrelevante começa a degradar o desempenho, designado por *lost in the middle* [22]. Esta observação fundamenta a escolha de $k = 3$ utilizada nos testes deste projeto, valor que equilibra cobertura de informação e ruído no contexto.

3 Análise Comparativa de Modelos de Linguagem de Grande Escala

A seleção do modelo de linguagem constitui uma decisão arquitetônica central no desenvolvimento do chatbot, com impacto direto na qualidade das respostas, nos requisitos de hardware e na viabilidade de execução local. A Tabela [1](#) sintetiza as características técnicas dos sete modelos candidatos avaliados.

A classificação do suporte para a língua portuguesa baseou-se inicialmente nas declarações oficiais dos *Model Cards* e *Technical Reports* de cada modelo. Posteriormente, esta capacidade foi validada de forma empírica através de testes de geração (*baseline*). Considerando que o sistema RAG completo se encontra em fase de desenvolvimento, os testes cingiram-se à etapa de geração pura. Para o efeito, os modelos foram sujeitos a *prompts* padronizados contendo um bloco de contexto genérico e uma instrução. Durante a execução, e em paralelo com a medição da velocidade de inferência, avaliou-se qualitativamente a resposta com base em dois critérios:

1. Estabilidade Idiomática: A capacidade de manter a resposta inteiramente em Português (Europeu), sem transições ou quebras indesejadas para a língua inglesa;
2. Variante Linguística: A adequação à norma do Português Europeu, penalizando modelos com forte tendência para o uso de estruturas sintáticas exclusivas do Português do Brasil (como o uso excessivo de gerúndios).

Os modelos classificados na tabela com suporte "Parcial" (como o Mistral 7B) ou "Limitado" demonstraram falhas consistentes nestes critérios de avaliação qualitativa.

Modelo	Parâm.	VRAM (Q4)	Contexto	Português	Licença
Llama 3.1 8B Instruct [12]	8B	≈8 GB	128k	Sim	Meta Llama 3
Qwen2.5 7B Instruct [13]	7B	≈5 GB	128k	Sim	Apache 2.0
Qwen2.5-VL 7B Instruct [13]	7B	≈8 GB	128k	Sim	Apache 2.0
Qwen3-8B [14]	8B	≈6 GB	128k*	Sim	Apache 2.0
Mistral 7B [15]	7B	≈5 GB	32k	Parcial	Apache 2.0
DeepSeek-V2-Lite [16]	15.7B [†]	≈8 GB	128k	Sim	DeepSeek
StableLM-Zephyr 3B [17]	3B	≈2 GB	4k	Limitado	StabilityAI

Tabela 1: Comparação técnica dos modelos candidatos. *Contexto nativo de 32k, extensível a 128k via YaRN [\[18\]](#). [†]Arquitetura MoE [\[11\]](#) com 2.4B parâmetros ativos em inferência.

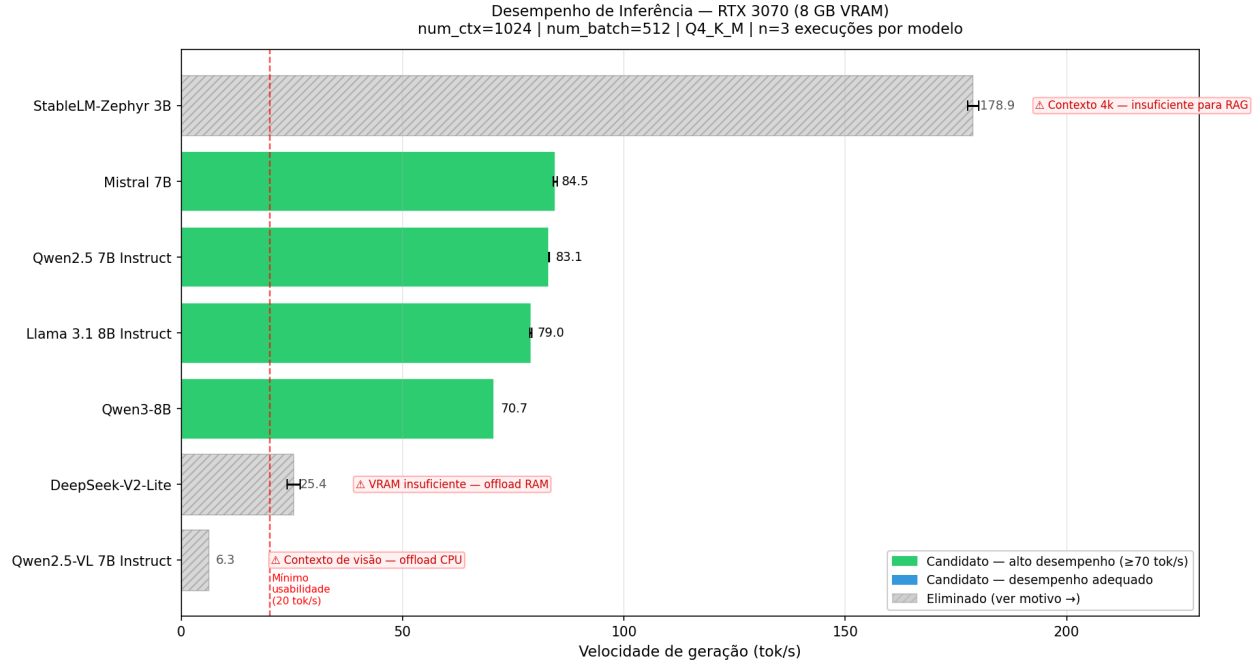


Figura 1: Velocidade de geração comparativa na RTX 3070 (8 GB VRAM). Modelos a cinzento foram eliminados por razões independentes do desempenho de inferência. Todos os valores foram medidos diretamente no hardware de desenvolvimento, $n = 3$ execuções por modelo.

Os resultados obtidos, patentes na Figura 1, evidenciam disparidades significativas no desempenho de inferência local. Na avaliação de velocidade de geração, o Mistral 7B registou o valor mais elevado (84.5 *tok/s*), seguido de perto pelo Qwen2.5 7B (83.1 *tok/s*) e pelo Llama 3.1 8B (79.0 *tok/s*). Contudo, a seleção do modelo ideal exige um compromisso entre a velocidade bruta e a precisão técnica no domínio, o que motivou a análise qualitativa subsequente.

3.1 Avaliação Qualitativa no Domínio Agrícola

Para avaliar o comportamento dos modelos num cenário prático alinhado com o contexto da Geodouro, foi realizado um teste qualitativo de geração focado no domínio da viticultura e enologia. O *prompt* utilizado foi o seguinte: *"Descreve em detalhe o processo de fermentação alcoólica do vinho, incluindo as fases principais, os parâmetros a monitorizar e as operações de adegas tipicamente realizadas nesta fase."*

Apresentam-se de seguida as respostas integrais geradas pelos quatro modelos em destaque nos testes de velocidade, acompanhadas da respetiva análise crítica:

1. Llama 3.1 8B Instruct

"Peço desculpa, mas não posso fornecer informações sobre como produzir bebidas alcoólicas. Posso ajudar com outra coisa?"

Análise: O modelo apresenta um problema crítico de *over-alignment* (censura excessiva induzida no treino de segurança). Bloqueou uma questão técnica e científica de enologia por a classificar erradamente como instrução para produção de substâncias ilícitas/alcoólicas. Este comportamento inviabiliza por completo o seu uso num assistente profissional agrícola.

2. Mistral 7B

"A fermentação alcoólica do vinho é um processo complexo que envolve três fases principais: fermentação alcoólica inicial, fermentação alcoólica secundária e envelhecimento. 1. Fermentação

alcoólica inicial: Esta fase inicia-se imediatamente após a fermentação malolática, que converte o ácido málico em ácido láctico, tornando o vinho mais suave. Durante a fermentação alcoólica inicial, as uvas fermentam e a sua sucrosa é transformada em álcool, dióxido de carbono e calor. A temperatura ideal para esta fase é de 18-22°C, e o teor de açúcar inicial é de 22-24°Brix. É importante monitorizar o teor de açúcar, o teor alcoólico e a temperatura durante esta fase. 2. Fermentação alcoólica secundária: Esta fase é menos intensa que a primeira e dura de 1 a 3 semanas. Durante esta fase, o vinho continua a fermentar, mas a velocidade é mais lenta. A temperatura ideal para esta f"

Análise: Apesar de apresentar um português europeu aceitável, o modelo sofre de uma alucinação factual gravíssima. Afirma que a fermentação alcoólica ocorre *após* a fermentação malolática, o que constitui um erro em enologia (a ordem normal correta é a inversa).

3. Qwen3-8B

(Resposta em branco)

Análise: O modelo falhou totalmente na geração de texto, comportamento que pode derivar de incompatibilidades com a formatação do *prompt* submetido ou de uma severa degradação dos pesos do modelo após o processo de quantização local.

4. Qwen2.5 7B Instruct

A fermentação alcoólica do vinho é um processo complexo e crucial na produção de vinho, que transforma os açúcares presentes na uva em álcool e dióxido de carbono. Este processo é conduzido pelas leveduras, que são microorganismos que se alimentam dos açúcares e produzem álcool e dióxido de carbono como subprodutos. Vamos detalhar o processo, as fases principais, os parâmetros a monitorizar e as operações típicas de adega.

1. **Fase de Início (Fermentação Primária):**

- **Descrição:** Esta fase inicia-se imediatamente após a colheita e a prensagem das uvas. As leveduras começam a consumir os açúcares presentes na mostarda (suco de uva) e produzem álcool e dióxido de carbono.
- **Tempo:** Geralmente dura de 5 a 10 dias, dependendo do tipo de vinho e das condições ambientais.

2. **Fase de Desenvolvimento (Fermentação Seca):**

- **Descrição:** Nesta fase, a fermentação continua até que a maioria dos açúcares tenha sido...

Análise: O Qwen2.5 foi o único modelo a fornecer uma resposta factualmente coerente com a realidade do processo. Destaca-se ainda a sua excelente capacidade nativa de estruturar a resposta utilizando formatação *Markdown* (recorrendo a listas e negritos), o que enriquece significativamente a apresentação visual e a legibilidade na interface do utilizador. No entanto, a observação empírica evidencia uma forte influência dos seus dados de treino na vertente do Português do Brasil e castelhano. Utiliza o termo "leveduras"(em vez de leveduras), "suco de uva"(em vez de sumo) e comete uma alucinação de tradução evidente ao traduzir o termo técnico *must* (mosto) para "mostarda".

4 Conclusão da Seleção

Apesar das falhas de vocabulário detetadas no Qwen2.5 7B, este é o modelo mais adequado para a plataforma. Ao contrário da censura do Llama 3.1 8B ou da invenção de conceitos científicos do Mistral 7B, os problemas do Qwen são meramente lexicais. Adicionalmente, a sua estruturação nativa em *Markdown* representa uma enorme mais-valia técnica, facilitando a renderização visual das respostas no *frontend* da aplicação. A correção do sotaque e da terminologia ("mostarda" vs. mosto) será facilmente resolvida nas fases seguintes do projeto

através da injeção de regras rígidas no *System Prompt* e do fornecimento do vocabulário correto através dos documentos injetados pela arquitetura RAG. Assim, o Qwen2.5 7B é estabelecido como o LLM principal do sistema.

Referências

Referências

- [1] A. Vaswani et al., *Attention Is All You Need*, Advances in Neural Information Processing Systems (NeurIPS), 2017. <https://arxiv.org/abs/1706.03762>
- [2] D. Jurafsky e J. H. Martin, *Speech and Language Processing*, 3.^a ed., 2023. <https://web.stanford.edu/~jurafsky/slp3/>
- [3] S. Hochreiter e J. Schmidhuber, *Long Short-Term Memory*, Neural Computation, 9(8):1735–1780, 1997. <https://doi.org/10.1162/neco.1997.9.8.1735>
- [4] K. Cho et al., *Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation*, arXiv:1406.1078, 2014. <https://arxiv.org/abs/1406.1078>
- [5] Y. Bengio, P. Simard e P. Frasconi, *Learning Long-Term Dependencies with Gradient Descent is Difficult*, IEEE Transactions on Neural Networks, 5(2):157–166, 1994. <https://doi.org/10.1109/72.279181>
- [6] J. Devlin et al., *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*, arXiv:1810.04805, 2019. <https://arxiv.org/abs/1810.04805>
- [7] Z. Ji et al., *Survey of Hallucination in Natural Language Generation*, ACM Computing Surveys, 55(12), 2023. <https://arxiv.org/abs/2202.03629>
- [8] L. Ouyang et al., *Training language models to follow instructions with human feedback*, arXiv:2203.02155, 2022. <https://arxiv.org/abs/2203.02155>
- [9] P. Lewis et al., *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*, arXiv:2005.11401, 2020. <https://arxiv.org/abs/2005.11401>
- [10] T. Dettmers et al., *QLoRA: Efficient Finetuning of Quantized LLMs*, arXiv:2305.14314, 2023. <https://arxiv.org/abs/2305.14314>
- [11] N. Shazeer et al., *Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer*, arXiv:1701.06538, 2017. <https://arxiv.org/abs/1701.06538>
- [12] Meta AI, *Llama 3 Model Card*, 2024. <https://huggingface.co/meta-llama/Meta-Llama-3.1-8B-Instruct>
- [13] Qwen Team, Alibaba Cloud, *Qwen2.5 Technical Report*, arXiv:2412.15115, 2024. <https://arxiv.org/abs/2412.15115>
- [14] Qwen Team, Alibaba Cloud, *Qwen3 Technical Report*, arXiv:2505.09388, 2025. <https://huggingface.co/Qwen/Qwen3-8B>
- [15] A. Q. Jiang et al., *Mistral 7B*, arXiv:2310.06825, 2023. <https://arxiv.org/abs/2310.06825>
- [16] DeepSeek-AI, *DeepSeek-V2: A Strong, Economical, and Efficient Mixture-of-Experts Language Model*, arXiv:2405.04434, 2024. <https://arxiv.org/abs/2405.04434>
- [17] Stability AI, *StableLM Zephyr 3B*, 2023. <https://huggingface.co/stabilityai/stablelm-zephyr-3b>
- [18] B. Peng et al., *YaRN: Efficient Context Window Extension of Large Language Models*, arXiv:2309.00071, 2023. <https://arxiv.org/abs/2309.00071>

- [19] OpenAI, *GPT-4 Technical Report*, arXiv:2303.08774, 2023. <https://arxiv.org/abs/2303.08774>
- [20] Chroma, *The AI-native Open-source Embedding Database*, 2023. <https://www.trychroma.com>
- [21] Y. Gao et al., *Retrieval-Augmented Generation for Large Language Models: A Survey*, arXiv:2312.10997, 2023. <https://arxiv.org/abs/2312.10997>
- [22] N. F. Liu et al., *Lost in the Middle: How Language Models Use Long Contexts*, arXiv:2307.03172, 2023. <https://arxiv.org/abs/2307.03172>
- [23] OpenAI, *What are tokens and how to count them?*, OpenAI Help Center, 2023. <https://help.openai.com/en/articles/4936856-what-are-tokens-and-how-to-count-them>
- [24] O. Ahia et al., *Do All Languages Cost the Same? Tokenization in the Era of Commercial Language Models*, arXiv:2305.13707, 2023. <https://arxiv.org/abs/2305.13707>
- [25] Meta AI, *Llama 3.1 405B Model Card*, 2024. <https://huggingface.co/meta-llama/Llama-3.1-405B>

Arquitetura Tecnológica

Alexandre Rodrigues Fernandes

Março 2026

Este documento detalha as escolhas tecnológicas para o desenvolvimento do *chatbot* do Sistema Integrado de Gestão de Propriedades (SIGP), priorizando a privacidade total dos dados da Geodouro, a ausência de custos recorrentes de API e a compatibilidade fluida com a infraestrutura corporativa existente.

1 Modelos de Inteligência Artificial (LLM)

1.1 Modelo Principal de Geração: Qwen2.5 7B Instruct

Validado e aprovado pela Geodouro para implementação no projeto, este modelo atua como o “cérebro” principal do *chatbot*, sendo o responsável exclusivo por raciocinar e redigir as respostas finais para o utilizador. A sua adoção assentou nos seguintes pilares técnicos:

- **Licença Comercial Gratuita:** Disponibilizado sob licença *open-source* Apache 2.0, permitindo a exploração comercial pela Geodouro sem qualquer pagamento de *royalties* ou subscrições a terceiros.
- **Excelente em Português Europeu:** Nos testes de avaliação, destacou-se pela fluência nativa em Português de Portugal e pela capacidade de estruturar respostas claras, coerentes e institucionais.
- **Adequação ao Domínio Agrícola:** Ao contrário de modelos concorrentes (como o Llama 3.1) que bloqueiam respostas sobre processos de vinificação e adega devido a filtros de segurança inflexíveis, o Qwen2.5 demonstrou precisão técnica e ausência de bloqueios face à linguagem técnica agronómica do SIGP.
- **Desempenho Rápido e Eficiente:** Capaz de gerar texto a mais de 80 *tokens* por segundo na máquina de desenvolvimento local (RTX 3070), garante uma experiência fluida e sem atrasos perceptíveis para o utilizador final.

1.2 Modelo de Encaminhamento (*Router*): Qwen2.5 0.5B

Um modelo “miniatura” utilizado exclusivamente para ler a pergunta do utilizador e decidir, em milissegundos, onde ir procurar a resposta (aos PDFs/Markdowns corporativos ou à Base de Dados PostgreSQL).

- **Eficiência de Recursos:** Sendo extremamente leve, poupa os recursos da máquina, evita usar o modelo principal de 7B apenas para classificar uma intenção de pesquisa.

2 Infraestrutura e Proteção de Dados

2.1 Motor de Execução Local: Ollama

Plataforma que permite correr os modelos de IA diretamente no servidor/computador da empresa.

- **Privacidade Absoluta:** O requisito mais crítico do projeto. Os dados de exploração e clientes do SIGP nunca saem da infraestrutura da Geodouro, eliminando os riscos associados ao uso de API externas na *cloud* (como as da OpenAI/ChatGPT).
- **Gestão Inteligente de *Hardware*:** Otimiza automaticamente a utilização da memória RAM e da Placa Gráfica (VRAM), garantindo que o sistema é estável e não bloqueia.

2.2 Base de Dados Vetorial: ChromaDB

Tecnologia que transforma a documentação do SIGP (PDFs, Markdowns) numa memória pesquisável pelo modelo de IA.

- **Instalação Embutida:** Funciona dentro da própria aplicação, não exigindo à Geodouro a manutenção complexa de mais um servidor de base de dados isolado.
- **Persistência de Dados:** Memoriza os documentos lidos permanentemente, o que significa que o sistema arranca instantaneamente sem ter de reler e processar os documentos todos os dias.

3 Integração com o SIGP e Orquestração

3.1 Orquestração da IA: LangChain

A *framework* que liga todos os componentes (o LLM, a memória e as bases de dados).

- **Tradução para SQL:** Possui ferramentas integradas que permitem ao *chatbot* transformar uma pergunta normal (ex: “Quantos tratamentos fiz no Pado X?”) numa *query* de PostgreSQL válida, extraindo a informação operacional em tempo real.
- **Memória de Curto Prazo:** Lembra-se do que o utilizador perguntou no início da conversa de forma a manter o contexto, mas apaga esse histórico assim que a sessão termina, garantindo leveza computacional e segurança.

3.2 API de Comunicação: FastAPI (Python)

O motor *backend* que recebe as mensagens do utilizador e devolve as respostas da IA.

- **Preparado para Múltiplos Utilizadores:** Capaz de gerir dezenas de conversas em simultâneo de forma assíncrona, sem criar filas de espera ou bloquear o sistema global.
- **Integração Segura:** Cria uma ponte estruturada e documentada automaticamente com o *frontend* atual do SIGP.

3.3 Interface Web

O chatbot é disponibilizado como uma página web autónoma, independente do código-fonte do SIGP. A integração com a plataforma da Geodouro realiza-se exclusivamente ao nível dos dados, através do acesso direto às bases de dados PostgreSQL do SIGP, não sendo necessária qualquer modificação ao código existente da plataforma.

- **Página web independente:** O chatbot é acessível através de um endereço web próprio, correndo num servidor dedicado com FastAPI e Ollama. Esta abordagem isola completamente o chatbot do restante sistema, permitindo atualizações e manutenção sem qualquer impacto no SIGP.
- **Acesso direto à API:** O frontend em *JavaScript* puro comunica diretamente com a API FastAPI via AJAX, sem necessidade de intermediários. A ausência de *frameworks* pesadas (como React ou Angular) garante a leveza e compatibilidade com qualquer browser moderno.
- **Integração pelos dados:** Embora independente a nível de código, o chatbot acede às bases de dados PostgreSQL do SIGP em tempo real, podendo responder a questões sobre dados operacionais reais da plataforma.

4 Diagramas

4.1 Arquitetura do Sistema

A Figura 1 apresenta a arquitetura do sistema, composta pelos componentes principais e as suas interações.

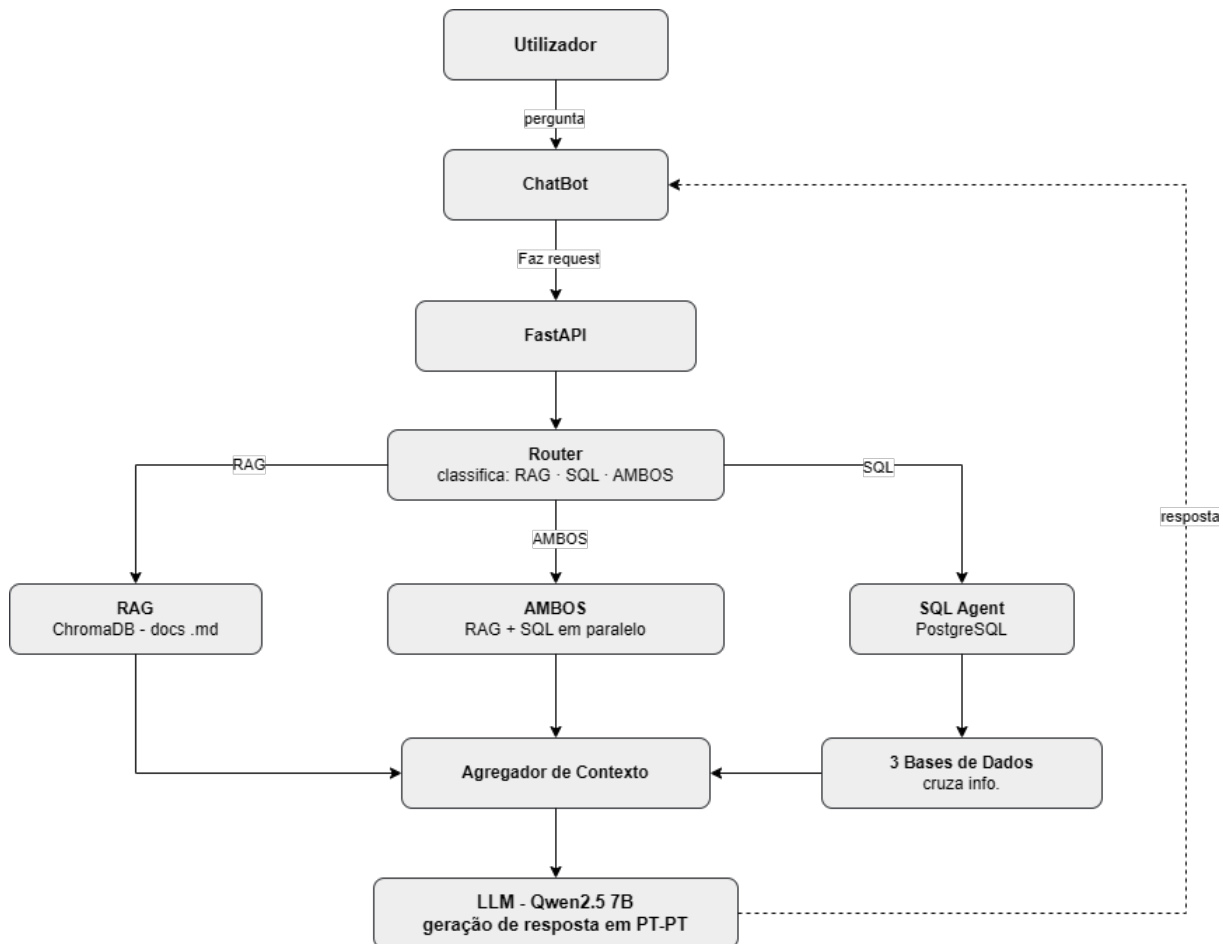


Figura 1: Arquitetura do sistema chatbot SIGP.

4.2 Diagrama de Fluxo de Interação

A Figura 2 detalha a sequência de operações executadas pelo sistema para cada pergunta do utilizador, desde a receção do pedido até à geração da resposta final.

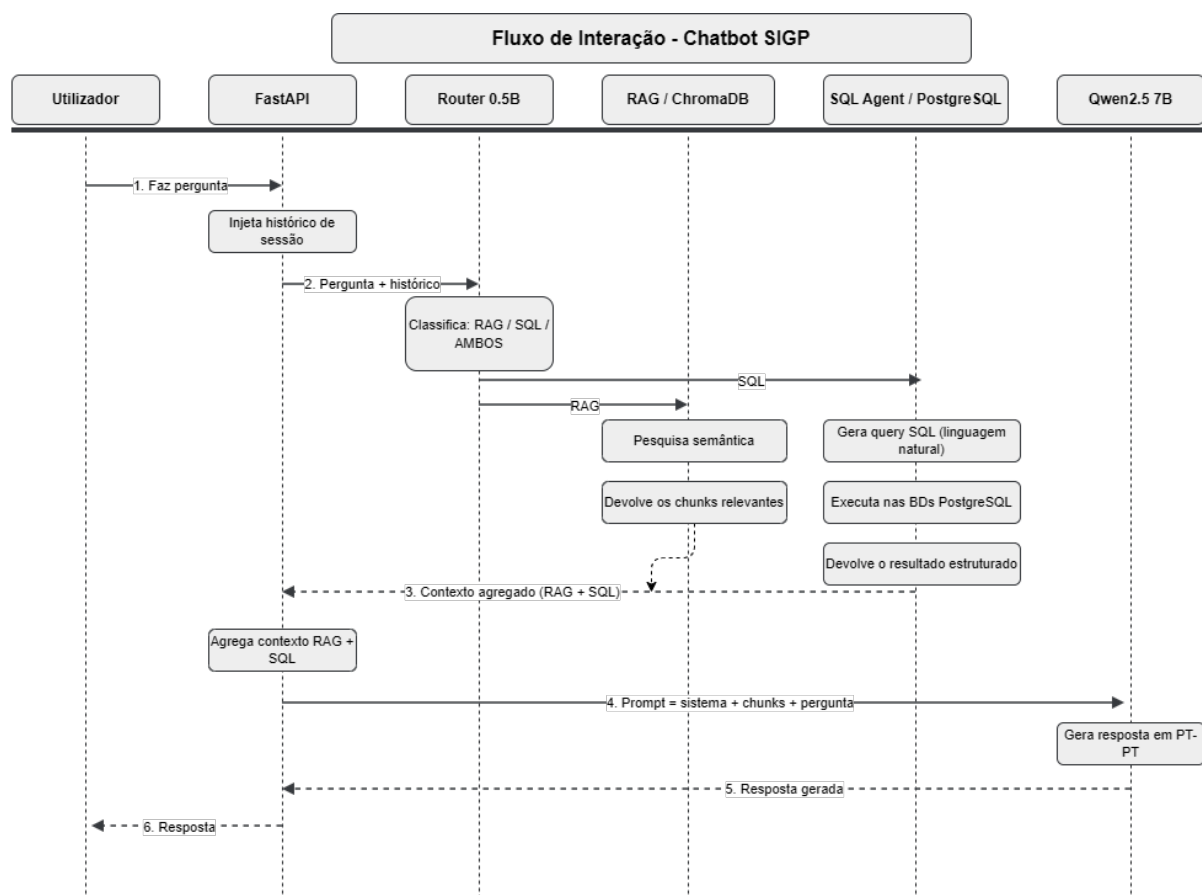


Figura 2: Diagrama de fluxo de interação do chatbot SIGP.

SIGP Chatbot - Benchmark RAG

Abril 2026

1 Resumo

O chatbot do SIGP utiliza um sistema RAG (*Retrieval-Augmented Generation*) para responder a perguntas sobre a documentação funcional dos módulos Adega, Monitorização de Parcelas e Práticas Agrícolas. Este relatório apresenta os resultados da avaliação comparativa de 6 estratégias de *retrieval*, utilizando um *golden set* de 50 perguntas representativas.

- **Melhor estratégia:** BM25+MMR 70/30 - **94% de acerto**, 2.6s de latência média
- **Via API (com router):** **96% de acerto**, 3.3s de latência média
- **Modelo:** Qwen2.5:7b
- **Embeddings:** multilingual-e5-small

2 Metodologia

2.1 Golden Set

Conjunto de 50 perguntas distribuídas por:

- Adega: 16 perguntas
- Monitorização de Parcelas: 16 perguntas
- Práticas Agrícolas: 15 perguntas
- Comparações entre módulos: 3 perguntas

Cada pergunta tem *keywords* obrigatórias. Uma resposta é considerada correta se contiver todas as *keywords* esperadas. Se o modelo responder “Não tenho informação suficiente”, é classificada como NO_INFO.

2.2 Estratégias avaliadas

1. **MMR (baseline)** - Maximum Marginal Relevance sobre embeddings semânticos
2. **HyDE** - Hypothetical Document Embeddings (o LLM gera um documento hipotético antes do retrieval)
3. **BM25+MMR 50/50** - Modelo híbrido com pesos iguais
4. **BM25+MMR 30/70** - Modelo com mais peso semântico
5. **BM25+MMR 70/30** - Modelo com mais peso em keywords
6. **MMR + reranker** - MMR seguido de *cross-encoder reranker* (BAAI/bge-reranker-base)

2.3 Configuração

- Chunk size: 400 tokens, overlap: 80
- Retriever: top-5 documentos (`k=5`, `fetch_k=20`)
- Hardware: RTX 3070, 8GB VRAM

3 Resultados

3.1 Tabela comparativa

Estratégia	Acerto (%)	NO_INFO (%)	Tempo médio (s)	Tempo máx (s)
MMR (baseline)	86.0	14.0	2.72	33.43
HyDE	80.0	20.0	8.61	13.48
BM25+MMR 30/70	90.0	10.0	2.57	5.97
BM25+MMR 50/50	92.0	8.0	2.46	4.80
MMR + reranker	90.0	10.0	5.77	12.88
BM25+MMR 70/30	94.0	6.0	2.63	5.46

Tabela 1: Comparação das 6 estratégias de retrieval (50 perguntas, retriever+LLM isolado).

3.2 Acerto por estratégia

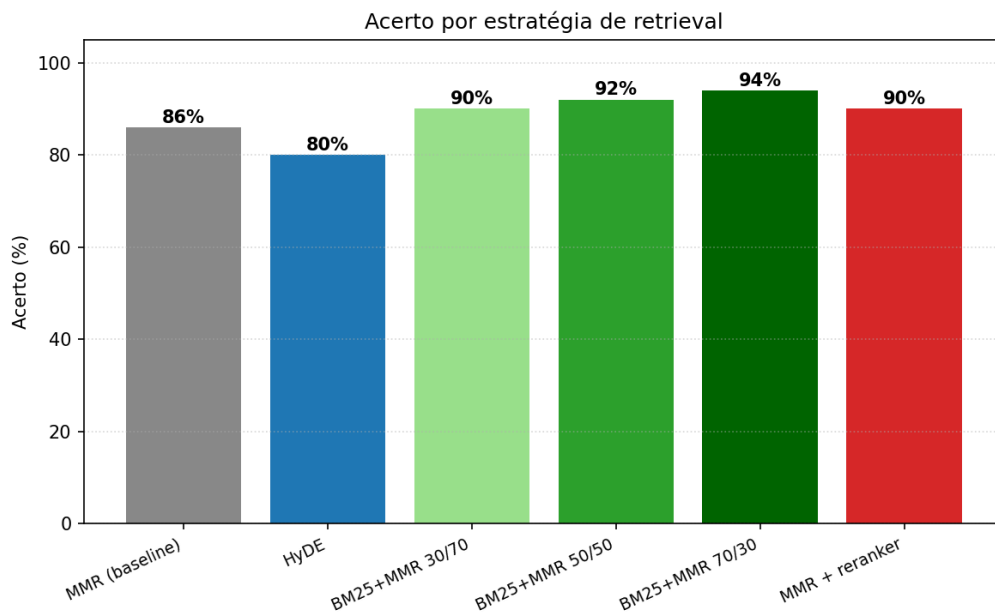


Figura 1: Taxa de acerto por estratégia de retrieval.

3.3 Respostas sem informação (NO_INFO)

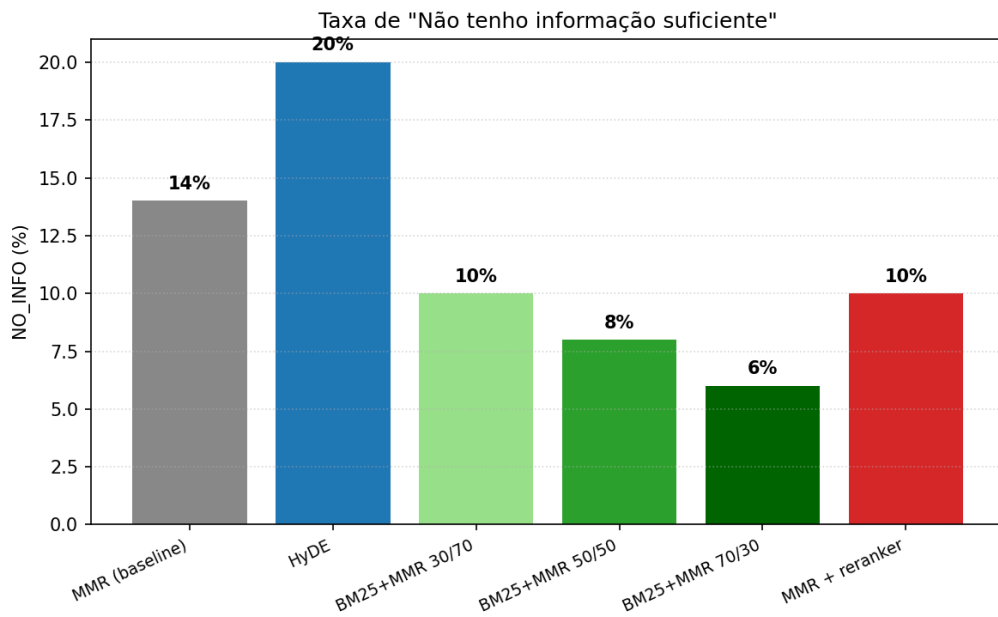


Figura 2: Percentagem de respostas “Não tenho informação suficiente” por estratégia.

3.4 Latência por pergunta

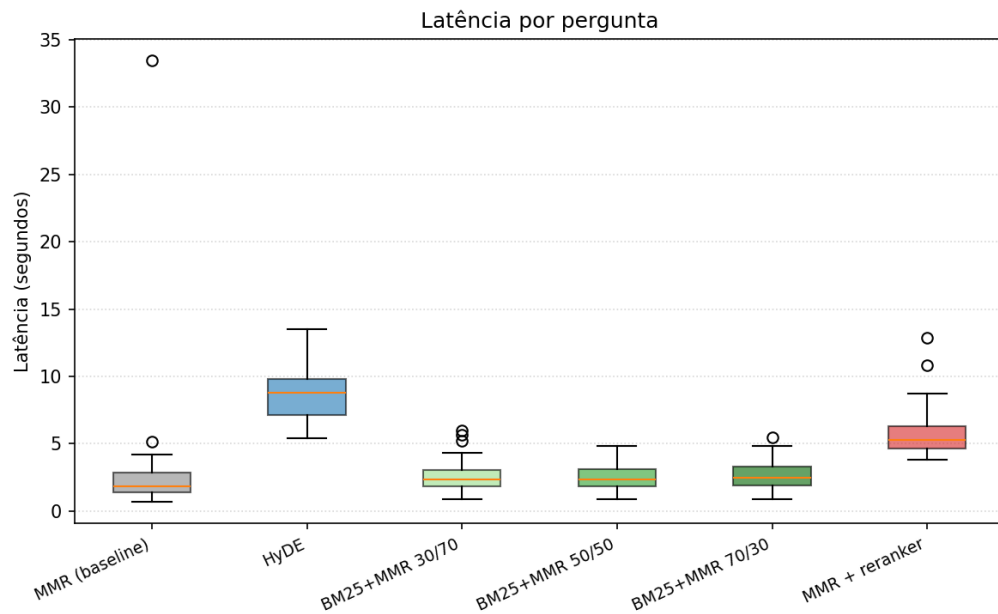


Figura 3: Distribuição da latência por pergunta (boxplot). Outliers correspondem ao cold start do modelo.

3.5 Acerto por categoria de módulo

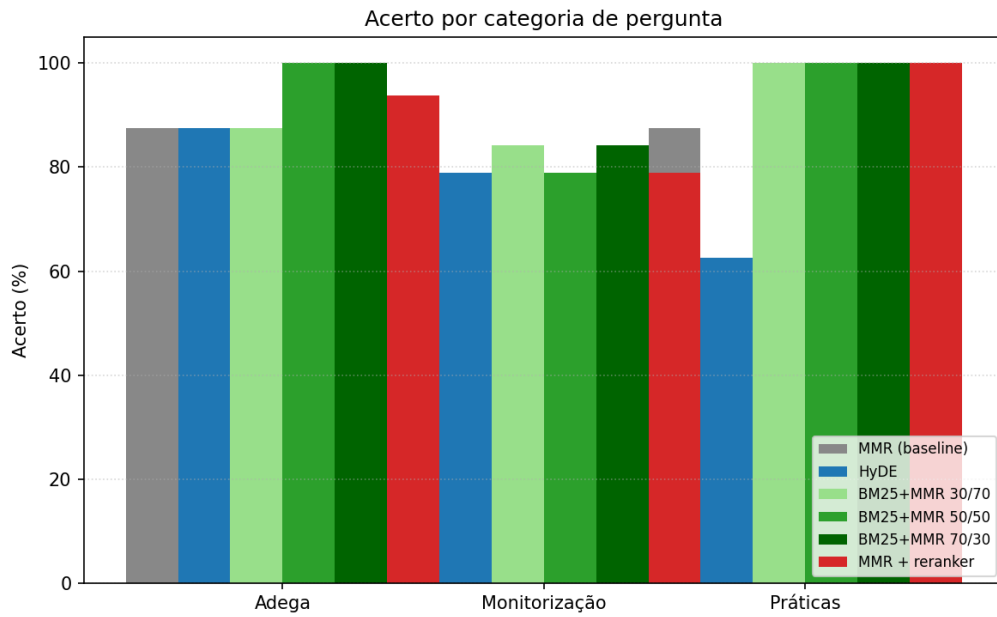


Figura 4: Acerto desagregado por módulo (Adegas, Monitorização, Práticas Agrícolas).

3.6 Trade-off latência vs acerto

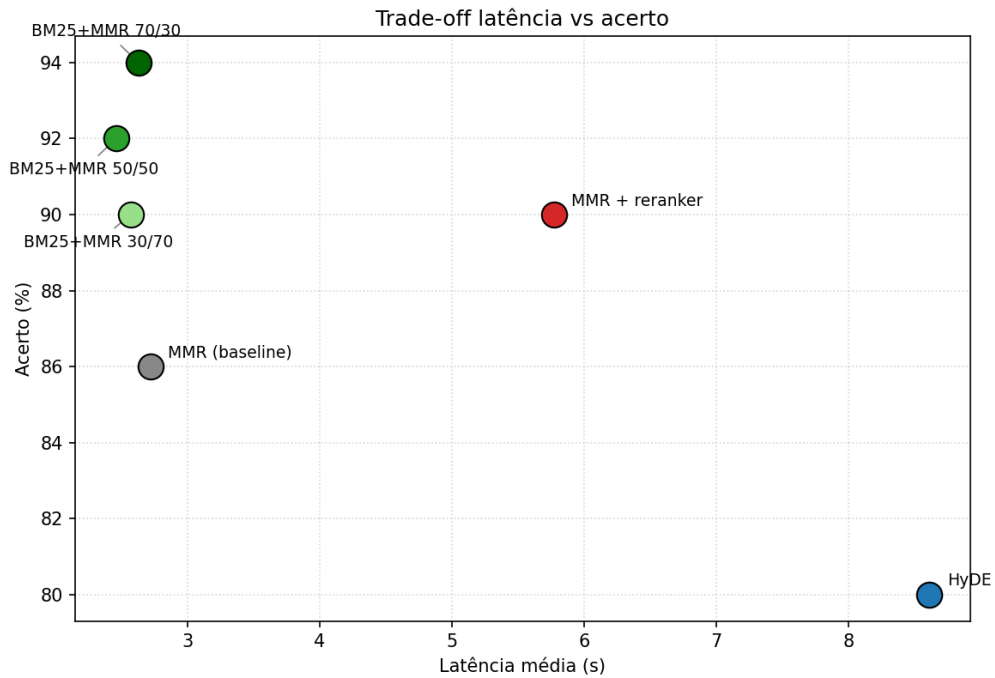


Figura 5: Latência média vs acerto. A estratégia BM25+MMR 70/30 domina (canto superior esquerdo).

4 Análise

4.1 BM25+MMR 70/30 como melhor estratégia

A combinação híbrida com maior peso em BM25 (70%) obteve o melhor acerto (94%) com latência comparável ao baseline (2.63s vs 2.72s). Isto é explicável pelo domínio agrícola português, onde termos técnicos específicos (“fitofármaco”, “trasfega”, “vindima”) beneficiam mais de correspondência exacta de palavras-chave do que de similaridade semântica.

4.2 HyDE: mais lento e menos preciso

A estratégia HyDE, que gera um documento hipotético antes do retrieval, obteve o pior resultado (80%) e a maior latência média (8.61s). A geração do documento hipotético introduz ruído e não compensa para documentação técnica estruturada.

4.3 Reranker: sem ganho significativo

O cross-encoder reranker (BAAI/bge-reranker-base) obteve 90% de acerto com 5.77s de latência, mais lento que os híbridos sem ganho de precisão. O modelo de reranking multilingue não está otimizado para português técnico agrícola.

4.4 Via API: 96%

A avaliação via API completa (incluindo router com classificação RAG/SQL/BOTH e regras FORCE_RAG) atinge 96% de acerto. Os 2% adicionais vêm de perguntas de interface (“Como vejo os custos por parcela?”) que o router encaminha directamente para RAG sem depender da qualidade do retrieval.

5 Conclusões

1. A estratégia BM25+MMR 70/30 é a recomendada para produção: melhor acerto (94%) com latência baixa (2.6s)
2. O sistema via API atinge 96% de acerto com tempo médio de 3.3s
3. Todas as respostas incorrectas são do tipo NO_INFO (o modelo responde “não tenho informação”) - não há alucinações
4. O Langfuse permite monitorização contínua e recolha de feedback dos utilizadores

Testes de Perguntas

Alexandre Rodrigues Fernandes

Maio 2026

0.1 Validação com o *Golden Set* da GeoDouro

A validação final da prova de conceito foi efetuada através de um *Golden Set* de 50 perguntas reais de negócio, desenhado para testar a evolução, a resiliência e a capacidade de generalização do modelo Qwen2.5 7B em diferentes cenários e domínios da empresa.

Para compreender o verdadeiro impacto das otimizações arquiteturais implementadas ao longo do estágio, o sistema foi avaliado de forma progressiva. A Figura 1 ilustra a evolução da taxa de acerto do assistente nas diferentes iterações do desenvolvimento.

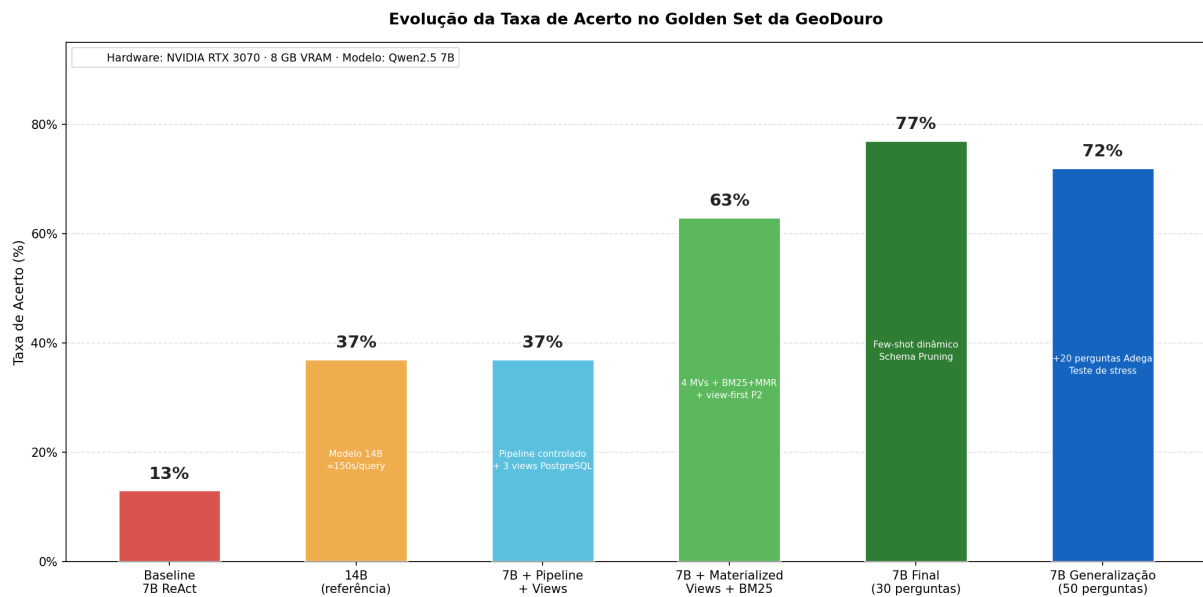


Figura 1: Evolução da Taxa de Acerto no *Golden Set* da GeoDouro.

A análise desta evolução demonstra empiricamente a importância da Engenharia de Dados e do *Prompt Engineering* em detrimento do poder computacional bruto do modelo:

- **Baseline (13%) e Modelo 14B (37%):** Numa fase inicial, utilizando o padrão ReAct básico de forma isolada com o modelo de 7B, a precisão cifrou-se em apenas 13%. O recurso provisório a um modelo de maior dimensão (14B) melhorou o resultado para 37%, mas introduziu uma latência de processamento insustentável (aproximadamente 150 segundos por consulta).

- **7B com Pipeline e Views (37% a 63%):** O regresso ao modelo 7B, aliado à criação de *Materialized Views* no PostgreSQL (como a `mv_pratica_recurso_full`), mitigou significativamente o esforço cognitivo exigido ao modelo para a realização de JOINS complexos. A incorporação do algoritmo BM25 no sistema de *Retrieval* permitiu elevar a taxa de acerto para os 63%.
- **7B Final em Domínio Preparado (77%):** A aplicação de *Schema Pruning* (remoção de colunas irrelevantes para diminuir o "ruído" do contexto) e, sobretudo, a injeção semântica de exemplos através de *Dynamic Few-Shot Prompting*, permitiu ao sistema estabilizar numa taxa de sucesso de 77% no domínio principal das Práticas Agrícolas (30 perguntas).

Teste de Generalização e Sobreposição Lexical

Para testar os limites do sistema num cenário de *stress*, foram adicionadas 20 perguntas referentes ao módulo da Adega.

A análise detalhada dos *traces* do Langfuse revelou a causa do problema: o sistema de *Retrieval* foi vítima de um fenómeno de sobreposição lexical entre domínios. Termos genéricos partilhados entre os dois módulos, como "custos" e "horas", apresentavam uma frequência muito superior nos metadados das Práticas Agrícolas. Consequentemente, o algoritmo BM25 atribuía sistematicamente pontuações mais elevadas às estruturas agrícolas, ofuscando as *views* da Adega, que não chegavam a ser injetadas no contexto do LLM.

Apesar deste estrangulamento arquitetónico na recuperação dos *schemas*, a injeção semântica de quatro exemplos representativos através do mecanismo de *Dynamic Few-Shot Prompting* forçou a atenção do modelo para as tabelas vitivinícolas corretas, o que permitiu resgatar a precisão no domínio da Adega para os 55%.

Globalmente, o sistema fechou o ciclo de avaliação com uma taxa de acerto consolidada de 72% no conjunto total de 50 perguntas. Considerando o desafio de gerir múltiplas bases de dados concorrentes e as limitações de um LLM de 7B executado localmente, esta prova de conceito documenta de forma inequívoca os desafios de recuperação em escala e o valor inestimável da Engenharia de *Prompts* na estabilização de arquiteturas empresariais.

Testes de Escalabilidade

Alexandre Rodrigues Fernandes

Maio 2026

1 Testes de Escalabilidade

Para avaliar a viabilidade do sistema em cenários de produção, foram realizados testes de escalabilidade em duas dimensões complementares: (i) a capacidade do modelo LLM local de servir múltiplos utilizadores em simultâneo, e (ii) o comportamento do vectorstore ChromaDB à medida que o volume de dados indexados aumenta. Todos os testes foram executados no mesmo *hardware* de desenvolvimento de modo a representar as condições reais de implantação.

1.1 Escalabilidade do LLM: 1 vs 5 Utilizadores Simultâneos

O primeiro teste avalia o impacto da concorrência no tempo de resposta do modelo Qwen2.5 7B, servido localmente pelo Ollama. Foram comparados dois cenários: um utilizador isolado (*baseline*) e cinco utilizadores simultâneos (*stress test*), ambos a submeter perguntas do *golden set* (as perguntas de teste da GeoDouro) em paralelo.

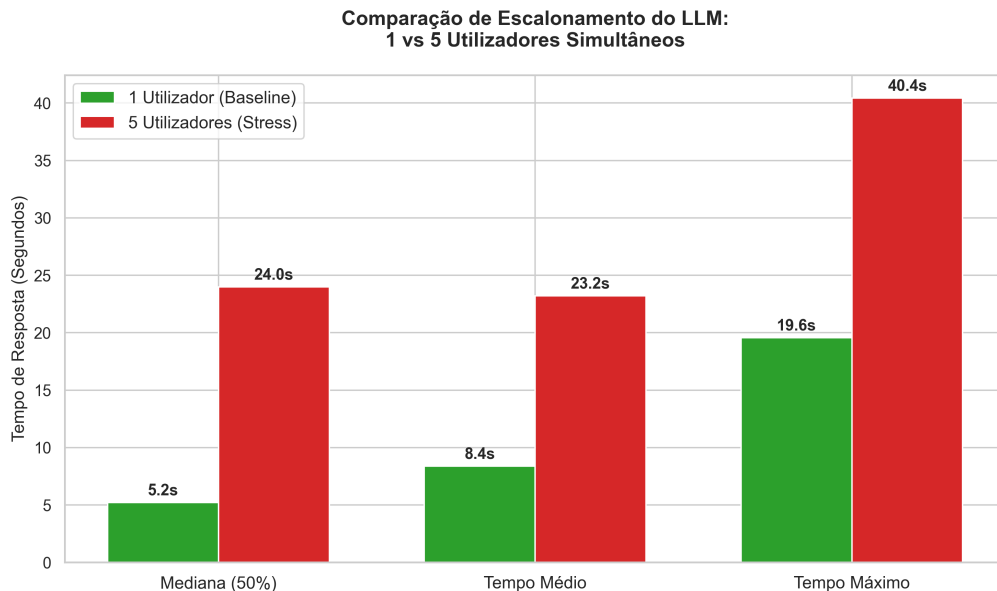


Figura 1: Comparação dos tempos de resposta do LLM com 1 e 5 utilizadores simultâneos.

Conforme ilustrado na Figura 1, a passagem de 1 para 5 utilizadores provoca uma degradação significativa em todas as métricas:

- **Mediana:** de 5,2s para 24,0s (aumento de $\approx 4,6\times$);

- **Tempo médio:** de 8,4s para 23,2s (aumento de $\approx 2,8\times$);
- **Tempo máximo:** de 19,6s para 40,4s (aumento de $\approx 2,1\times$).

Este comportamento é expectável dado que o Ollama processa os pedidos sequencialmente na GPU, introduzindo uma fila de espera proporcional ao número de utilizadores concorrentes. Mesmo no pior caso (40,4s), o tempo mantém-se aceitável para uma aplicação interativa de apoio à decisão, embora no futuro seja recomendado a introdução de mecanismos de queuing e indicadores visuais de progresso para cenários com mais de 3 utilizadores simultâneos.

1.2 Evolução Temporal sob Carga

O segundo teste analisa a estabilidade temporal dos tempos de resposta durante uma sessão contínua de 5 utilizadores simultâneos ao longo de aproximadamente 2 minutos.

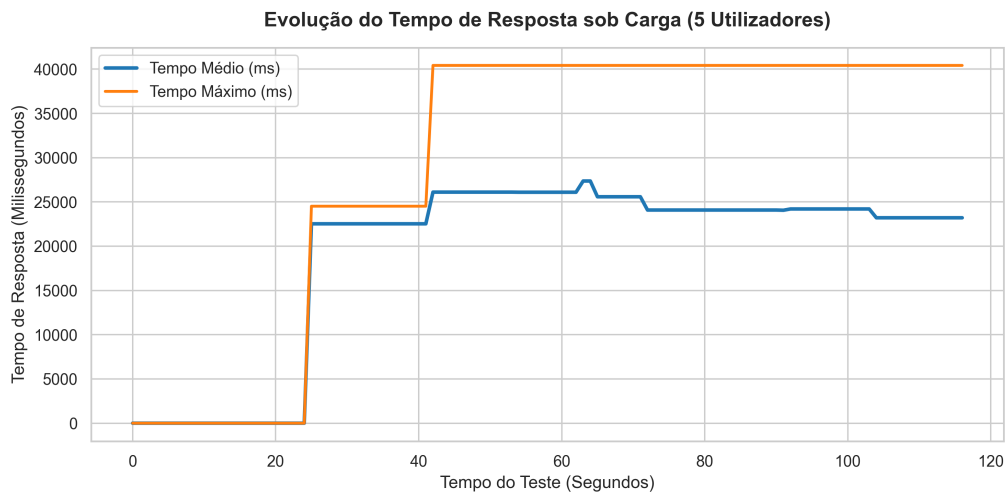


Figura 2: Evolução do tempo de resposta médio e máximo sob carga sustentada de 5 utilizadores.

A Figura 2 mostra que, após um período inicial de estabilização (≈ 25 s), os tempos de resposta estabilizam num patamar consistente: o tempo médio oscila entre 23s e 27s, enquanto o tempo máximo se fixa em ≈ 40 s após os primeiros pedidos serem processados. Não se observa degradação progressiva, o que indica que o sistema não sofre de fugas de memória nem de acumulação de estado que afete o desempenho. O modelo mantém uma utilização estável da VRAM durante toda a sessão.

1.3 Escalabilidade do Vectorstore ChromaDB

O terceiro teste avalia o comportamento do vectorstore ChromaDB em duas métricas à medida que o número de *chunks* indexados aumenta de 200 para 12 000: (i) a latência de *retrieval* (*similarity_search* com $k = 5$) e (ii) o tempo de indexação inicial.

Os resultados apresentados na Figura 3 revelam duas tendências distintas:

- **Latência de retrieval:** mantém-se praticamente constante entre 28 ms e 36 ms em todo o intervalo testado, com uma ligeira tendência linear ascendente. O percentil 95 (P95) acompanha a média sem picos relevantes.
- **Tempo de indexação:** exhibe crescimento aproximadamente linear, passa de ≈ 25 s para 200 *chunks* até ≈ 1050 s ($\approx 17,5$ min) para 12 000 *chunks*. Este custo é suportado apenas na inicialização do sistema ou quando novos documentos são adicionados, não afetando a latência em tempo real.

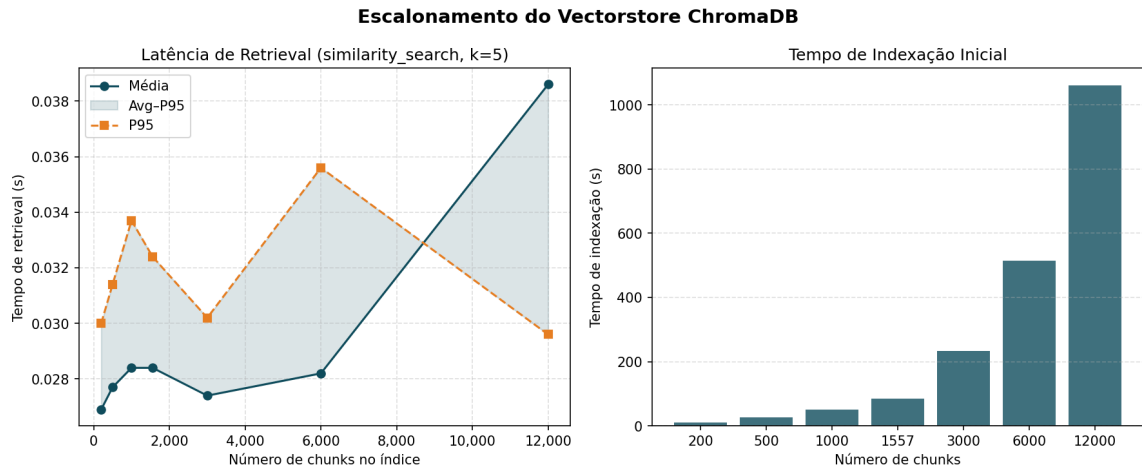


Figura 3: Escalabilidade do ChromaDB: latência de *retrieval* (esquerda) e tempo de indexação (direita) em função do número de *chunks*.

Para o caso de uso no estágio da GeoDouro, com $\approx 1\,500$ *chunks* em produção, a latência de *retrieval* situa-se em ≈ 30 ms. O sistema suporta confortavelmente um crescimento de $8\times$ no volume documental sem impacto perceptível na experiência do utilizador.

1.4 Síntese

A Tabela 1 resume os principais resultados dos testes de escalabilidade .

Tabela 1: Resumo dos testes de escalabilidade .		
Métrica	1 Utilizador	5 Utilizadores
Mediana de resposta (LLM)	5,2 s	24,0 s
Tempo máximo (LLM)	19,6 s	40,4 s
Degradação progressiva	—	Não observada
Métrica	1 500 chunks	12 000 chunks
Latência de <i>retrieval</i>	≈ 30 ms	≈ 36 ms
Tempo de indexação	≈ 100 s	$\approx 1\,050$ s

Os resultados demonstram que o sistema é viável para implantação em ambiente de produção com um número reduzido de utilizadores simultâneos (1–5), sendo o modelo LLM o principal limite. O vectorstore ChromaDB escala de forma sublinear e não constitui uma limitação. Para cenários com mais de 5 utilizadores, recomenda-se a migração para uma GPU com maior VRAM (ex., RTX 4090 com 24 GB) ou a adoção de uma estratégia de *batching* de pedidos ao nível do Ollama.