



UNIVERSIDADE FEDERAL DO MARANHÃO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
COORDENAÇÃO DE CIÊNCIA DA COMPUTAÇÃO
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

LUIS MIGUEL ALVES MEIRELES

**ARQUITETURA SERVERLESS PARA SISTEMAS MULTIAGENTES DE IA COM
OBSERVABILIDADE: UM ESTUDO DE CASO EM UMA CLÍNICA
ODONTOLÓGICA**

SÃO LUIS – MA
2026



LUIS MIGUEL ALVES MEIRELES

**ARQUITETURA SERVERLESS PARA ANÁLISE DE SISTEMAS MULTI-AGENTES
DE IA: UM PIPELINE DE DADOS ORIENTADO A EVENTOS PARA
AUTOMAÇÃO MULTICANAL E ANALYTICS**

Trabalho de Conclusão de Curso apresentado ao
Curso de Bacharelado em Ciência da
Computação da Universidade Federal do
Maranhão Campus São Luis, como requisito
parcial para obtenção do grau de Bacharel em
Ciência da Computação.

Orientador: Prof. Dr Mario Antônio Meireles
Teixeira

SÃO LUIS - MA
2026

Ficha gerada por meio do SIGAA/Biblioteca com dados fornecidos pelo(a) autor(a).
Diretoria Integrada de Bibliotecas/UFMA

Alves Meireles, Luis Miguel.

ARQUITETURA SERVERLESS PARA SISTEMAS MULTIAGENTES DE IA
COM OBSERVABILIDADE: UM ESTUDO DE CASO EM UMA CLÍNICA
ODONTOLÓGICA / Luis Miguel Alves Meireles. - 2025.

97 p.

Orientador(a): Mário Antônio Meireles Teixeira.

Curso de Ciência da Computação, Universidade Federal do
Maranhão, São Luís, 2025.

1. Ia. 2. Odontologia. 3. Dados. 4. Multiagentes.
5. Serverless. I. Meireles Teixeira, Mário Antônio. II.
Título.

LUIS MIGUEL ALVES MEIRELES

**ARQUITETURA SERVERLESS PARA ANÁLISE DE SISTEMAS MULTI-AGENTES
DE IA: UM PIPELINE DE DADOS ORIENTADO A EVENTOS PARA
AUTOMAÇÃO MULTICANAL E ANALYTICS**

Trabalho de Conclusão de Curso apresentado ao
Curso de Bacharelado em Ciência da
Computação da Universidade Federal do
Maranhão Campus São Luis, como requisito
parcial para obtenção do grau de Bacharel em
Ciência da Computação.

Aprovado em _____ de _____ de _____.

Banca Examinadora

Orientador

Prof. Dr Mário Antônio Meireles Teixeira
Universidade Federal do Maranhão

Membro da Banca

Prof. Dr Tiago Bonini Borchatt
Universidade Federal do Maranhão

Membro da Banca

Prof. Me Carlos Eduardo Portela Serra de Castro
Universidade Federal do Maranhão

AGRADECIMENTOS

Primeiramente, expresso minha gratidão a Deus, sem quem nada disso seria possível. Graças a Ele, cheguei aonde sempre almejei e encontro forças para, a cada dia, lutar na busca de um mundo melhor.

Gostaria de agradecer aos meus pais, Lourenço e Flediana, as pessoas que mais amo no mundo. Graças a eles fui capaz de chegar até aqui — são o alicerce da minha vida e fizeram tudo para que eu estivesse hoje escrevendo este trabalho. Os esforços de uma vida inteira jamais serão esquecidos. Agradeço ao meu irmão por ter me proporcionado meu primeiro curso de computação, a partir do qual nasceu o afeto pela área; sem dúvida nenhuma, sem ele eu não estaria aqui. Agradeço também à minha tia Fledilana, que em muitos momentos da minha vida foi minha segunda mãe e confidente.

Agradeço imensamente a todos os professores do Departamento de Ciência da Computação da Universidade Federal do Maranhão pela oportunidade de aprender muito — não só no aspecto acadêmico, mas no aspecto humano. Saio da UFMA uma pessoa muito melhor, e sei que o sou graças àqueles que me guiaram no processo de conhecer e de me conhecer. Um agradecimento especial ao meu orientador, o Professor Dr. Mário Antônio Meireles Teixeira. Sempre tivemos conversas riquíssimas sobre futuro e sobre vida — um ser humano admirável que me orientou desde que nos conhecemos, sendo verdadeiro mentor. Levarei comigo tudo que aprendi com ele ao longo desses anos.

Por fim, agradeço aos meus amigos Luís Felipe, João Lucas e Matheus Gutemberg, que conheci na faculdade e que desde cedo me acompanharam nessa trajetória acadêmica, tornando-a mais divertida e desafiadora. Sem dúvidas, chego onde estou graças à presença deles. Agradeço também às minhas amigas que tive o prazer de conhecer no trabalho e que hoje levo no peito como parte importante de quem sou: Camila Ramos, Lavínia Souza, Rafaela Amaral e Jeniffer Ainett, que ao longo da minha trajetória profissional sempre me apoiaram e incentivaram.

Aqui a virtude falhou em minha elevada fantasia, mas já movia o meu desejo e minha vontade, como uma roda girando em perfeita harmonia. O amor que move o sol e todas as estrelas.

Dante Alighieri.

RESUMO

A adoção de sistemas conversacionais baseados em inteligência artificial por pequenas e médias organizações ainda enfrenta desafios relacionados à escalabilidade, à observabilidade e ao custo operacional. Este trabalho propõe uma arquitetura serverless orientada a eventos para sustentar sistemas multiagentes de IA, desacoplando o processamento das interações da persistência e da análise dos dados. A solução foi projetada, implantada e avaliada empiricamente em uma clínica odontológica de estética avançada situada em Curitiba, por meio de um pipeline composto por Amazon S3, AWS Lambda e Amazon Athena, responsável por registrar e analisar as interações entre usuários e um conjunto de agentes especializados coordenados por um agente orquestrador. A avaliação empírica, conduzida sobre 46 interações reais registradas ao longo de 23 dias, demonstrou latência média de 22,1 segundos para interações textuais, identificou divergências entre planejamento e execução em 21,7% das interações e alcançou 91,7% de cobertura na auditoria automática das respostas, indicadores obtidos exclusivamente por meio de consultas SQL ao data warehouse, sem acesso ao código-fonte ou à infraestrutura de execução. O custo operacional apurado foi de R\$ 0,28 por interação, com custo praticamente nulo na camada analítica. Os resultados evidenciam a funcionalidade, a observabilidade e a viabilidade econômica da arquitetura proposta, indicando que a solução é tecnicamente viável e adequada para a adoção de sistemas conversacionais baseados em IA por pequenas e médias organizações.

Palavras-chave: Sistema multiagente. Arquitetura serverless. Arquitetura orientada a eventos. Observabilidade. Inteligência artificial conversacional..

ABSTRACT

The adoption of AI-based conversational systems by small and medium-sized organizations still faces challenges related to scalability, observability, and operational cost. This work proposes an event-driven serverless architecture to support AI multi-agent systems, decoupling interaction processing from data persistence and analysis. The solution was designed, deployed, and empirically evaluated at an advanced aesthetic dental clinic located in Curitiba, through a pipeline composed of Amazon S3, AWS Lambda, and Amazon Athena, responsible for recording and analyzing interactions between users and a set of specialized agents coordinated by an orchestrator agent. The empirical evaluation, conducted over 46 real interactions recorded over 23 days, showed an average latency of 22.1 seconds for text-based interactions, identified divergences between planning and execution in 21.7% of interactions, and achieved 91.7% coverage in the automated auditing of responses — indicators obtained exclusively through SQL queries to the data warehouse, without access to the source code or execution infrastructure. The measured operational cost was R\$ 0.28 per interaction, with a virtually negligible cost at the analytical layer. The results evidence the functionality, observability, and economic viability of the proposed architecture, indicating that the solution is technically feasible and suitable for the adoption of AI-based conversational systems by small and medium-sized organizations.

Keywords: Multi-agent system. Serverless architecture. Event-driven architecture. Observability. Conversational artificial intelligence

LISTA DE ILUSTRAÇÃO

Figura 3.1 – Fluxo do processo do sistema.	35
Figura 3.2 – Arquitetura Técnica do Sistema.	36
Figura 3.3 – Exemplo real de payload.	42
Figura 3.4 – Fluxograma de Execução da função Lambda.	60
Quadro 3.1 – Schema JSON dos campos enviados na requisição de entrada.	39
Quadro 3.2 – Schema do objeto textual no array input.	40
Quadro 3.3 – Schema do objeto de imagem no array input.	41
Quadro 3.4 – Agentes especialistas: domínio, usuário primário e tipo de tarefa	46
Quadro 3.5 – Dimensões de avaliação do Agente H e seus critérios.	48
Quadro 3.6 – Atribuição de modelos.....	50
Quadro 3.7 – Campos do bloco output.....	54
Quadro 3.8 – Campos de cada elemento do array steps.	55
Quadro 3.9 – Metadados de sessão.....	56
Quadro 3.10 – Estrutura da tabela messages.	62
Quadro 3.11 – Estrutura da tabela agents_plan.	64
Quadro 3.12 – Estrutura da tabela agents_actions.....	66
Quadro 3.13 – Métricas analíticas: definição, origem e proposição correspondente.	72
Quadro 4.1 – Proposições da tese.	87

LISTA DE TABELAS

Tabela 4.1 – Distribuição das interações por perfil identificado.....	77
Tabela 4.2 – Distribuição diária das interações no período de coleta.	78
Tabela 4.3 – Latência de resposta por tipo de entrada.....	80
Tabela 4.4 – Taxa de acionamento por agente especialista.....	81
Tabela 4.5 – Interações com divergência entre plano e execução.....	82
Tabela 4.6 – Pontuações médias por dimensão avaliada pelo Agente H	84
Tabela 4.7 – Custo de inferência por modelo no período.....	85

LISTA DE ABREVIATURAS E SIGLAS

API	Interface de Programação de Aplicações (Application Programming Interface)
AWS	Amazon Web Services
BI	Inteligência de Negócios (Business Intelligence)
CAD/CAM	Desenho Assistido por Computador / Manufatura Assistida por Computador (Computer-Aided Design / Computer-Aided Manufacturing)
EDA	Arquitetura Orientada a Eventos (Event-Driven Architecture)
ETL	Extração, Transformação e Carga (Extract, Transform, Load)
GPT	Transformador Generativo Pré-treinado (Generative Pre-trained Transformer)
HTTP	Protocolo de Transferência de Hipertexto (Hypertext Transfer Protocol)
IA	Inteligência Artificial
ISO 8601	Norma internacional de representação de data e hora
JSON	Notação de Objetos JavaScript (JavaScript Object Notation)
LGPD	Lei Geral de Proteção de Dados Pessoais
LLM	Modelo de Linguagem de Grande Escala (Large Language Model)
MAS	Sistema Multiagente (Multi-Agent System)
MIME	Extensões Multipropósito para Correio de Internet (Multipurpose Internet Mail Extensions)
OLAP	Processamento Analítico On-line (Online Analytical Processing)
OLTP	Processamento de Transações On-line (Online Transaction Processing)
OWASP	Projeto Aberto de Segurança em Aplicações Web (Open Web Application Security Project)
PDF	Formato de Documento Portátil (Portable Document Format)
PME	Pequena e Média Empresa
RAG	Geração Aumentada por Recuperação (Retrieval-Augmented Generation)
RLHF	Aprendizado por Reforço com Feedback Humano (Reinforcement Learning from Human Feedback)
RNN	Rede Neural Recorrente (Recurrent Neural Network)
S3	Amazon Simple Storage Service

SoC	Separação de Responsabilidades (Separation of Concerns)
SQL	Linguagem de Consulta Estruturada (Structured Query Language)
TTL	Tempo de Vida (Time to Live)
URL	Localizador Uniforme de Recursos (Uniform Resource Locator)
UTC	Tempo Universal Coordenado (Coordinated Universal Time)
VPS	Servidor Virtual Privado (Virtual Private Server)
BaaS	Backend as a Service
CLIP	Contrastive Language–Image Pretraining
CNN	Convolutional Neural Network
CoT	Chain-of-Thought
DW	Data Warehouse
FaaS	Functions as a Service
IETF	Internet Engineering Task Force
LSTM	Long Short-Term Memory
MA	Maranhão
P&D	Pesquisa e Desenvolvimento
PR	Paraná
RDS	Relational Database Service
ReAct	Reasoning + Acting
SIGAA	Sistema Integrado de Gestão de Atividades Acadêmicas
UFMA	Universidade Federal do Maranhão

SUMÁRIO

1	INTRODUÇÃO	15
2	REFERENCIAL TEÓRICO	18
2.1	Modelos de Linguagem de Grande Escala (LLMs)	18
2.1.1	Arquitetura Transformer e o Mecanismo de Atenção	18
2.1.2	Do Pré-treinamento ao Ajuste Fino	18
2.1.3	LLMs Multimodais: Integração de Texto e Imagem.....	19
2.1.4	Engenharia de Prompts e Chain-of-Thought	20
2.2	Sistemas Multi-Agentes de IA (MAS)	21
2.2.1	Definição e Propriedades de Agentes Inteligentes	21
2.2.2	Taxonomias de Arquiteturas Multi-Agentes	22
2.2.3	O Padrão Orquestrador-Trabalhador	22
2.2.4	Frameworks de Raciocínio e Ação: ReAct, Reflexion, AutoGen e LangGraph	23
2.2.5	Memória e Gestão de Contexto em Agentes LLM.....	24
2.3	Princípios Arquiteturais de Software para Sistemas de IA	25
2.3.1	Separação de Responsabilidades (SoC).....	25
2.3.2	Arquitetura Orientada a Eventos (EDA)	26
2.3.3	Computação Serverless.....	27
2.4	Pipelines de Dados: Do Data Lake ao Lakehouse Analítico	28
2.4.1	Data Lake: Conceito e Desafios	28
2.4.2	ETL Orientado a Eventos	29
2.4.3	O Padrão Lakehouse e a Consulta Analítica com S3 + Athena	29
2.4.4	Particionamento Hive-Style.....	30
2.5	IA Aplicada ao Setor de Saúde e Odontologia	31
2.5.1	Transformação Digital em Organizações de Pequeno e Médio Porte	31
2.5.2	Chatbots e Assistentes Virtuais em Contextos Clínicos.....	31
2.6	Lei Geral de Proteção de Dados (LGPD)	32
3	METODOLOGIA	33
3.1	Visão Geral e Princípios Arquiteturais	33
3.1.1	Separação de Responsabilidades como Princípio Central.....	33
3.1.2	Arquitetura Orientada a Eventos	33
3.1.3	Justificativa da Abordagem Serverless.....	34
3.2	Visão em Camadas do Sistema	34

3.2.1 Diagrama Geral da Arquitetura	35
3.2.2 Descrição das Cinco Camadas.....	36
3.3 Camada de Ingestão	37
3.3.1 Contrato de Dados de Entrada (Schema JSON)	38
3.3.2 Autenticação via Bearer Token	42
3.3.3 Pré-processamento Multimodal — Separação de Texto e Imagem	43
3.4 Camada de Inteligência — Sistema Multi-Agentes	44
3.4.1 Padrão Orchestrator-Workers aplicado	44
3.4.2 O Orquestrador — classificação de intenção e memória de sessão	45
3.4.3 Roteamento por modalidade — Agente A	45
3.4.4 Os agentes especialistas (B a G) — domínios e responsabilidades.....	45
3.4.5 O loop de self-reflection — Agente H e o schema de avaliação	47
3.4.6 Gestão de modelos de linguagem via OpenRouter.....	49
3.5 Camada de Persistência — Data Lake	51
3.5.1 Desacoplamento entre inferência e persistência	51
3.5.2 Organização do S3	52
3.5.3 Schema do JSON bruto persistido	53
3.6 Camada de Processamento — ETL Serverless	57
3.6.1 Trigger automático via evento S3	57
3.6.2 As três tabelas de destino e suas transformações	59
3.6.3 Particionamento Hive-style para o Athena	69
3.7 Camada Analítica — Data Warehouse	70
3.7.1 Amazon Athena como motor de consulta serverless.....	70
3.7.2 Métricas analisáveis e justificativa de cada uma	71
3.7.3 Modelo de dados e orientação para análise futura.....	73
3.8 Decisões de Design e Trade-offs	74
3.8.1 Por que n8n como orquestrador de fluxo?	74
3.8.2 Por que S3 + Athena e não um banco de dados relacional?	74
3.8.3 Limitações conhecidas da arquitetura proposta	75
4 RESULTADOS	77
4.1 Contexto da Implantação	77
4.2 Funcionalidade do Sistema	78
4.2.1 Volume de uso e distribuição temporal	78
4.2.2 Latência por tipo de input: texto versus multimodal	79

4.3 Observabilidade do Sistema	80
4.3.1 Taxa de acionamento por agente especialista	81
4.3.2 Análise de divergência entre plano e execução	82
4.3.3 Avaliação qualitativa via Agente H.....	83
4.4 Viabilidade Econômica	85
4.4.1 Custo total de inferência no período.....	85
4.4.2 Custo da camada analítica (Athena)	86
4.4.3 Síntese de viabilidade para PMEs	86
4.5 Síntese dos Resultados	87
5 CONCLUSÃO	90

1 INTRODUÇÃO

A transformação digital do setor odontológico consolidou-se, na última década, em torno de tecnologias de imagem e manufatura, radiografia digital, escaneamento intraoral e fluxos CAD/CAM, deslocando para o meio computacional etapas antes inteiramente manuais. Esse avanço, contudo, concentrou-se na dimensão técnica do ato clínico e deixou relativamente intocada a operação cotidiana das clínicas, que continua a articular, de forma simultânea, domínios heterogêneos: o atendimento clínico, a relação comercial com pacientes, a rotina técnica do laboratório e a produção de documentação jurídica. Uma clínica de pequeno porte opera todos esses domínios com equipe reduzida, o que torna o suporte a decisões e a padronização de respostas um gargalo operacional recorrente.

Paralelamente, a emergência dos foundation models reconfigurou as condições de acesso à inteligência artificial. Bommasani et al. (2021) descrevem esses modelos como uma ruptura na lógica de adoção: ao encapsularem conhecimento de propósito geral durante o pré-treinamento, podem ser especializados para domínios específicos por meio de instruções textuais, sem retreinamento nem infraestrutura própria. A barreira de infraestrutura, por sua vez, é endereçada pelo modelo serverless, que converte o custo fixo de servidores dedicados em custo estritamente variável e proporcional ao uso (WANG et al., 2018). A conjunção desses dois fatores torna, ao menos em princípio, tecnicamente viável que uma organização de pequeno porte opere agentes de IA especializados, hipótese que este trabalho examina em condições reais de operação, tomando como ambiente de implantação uma clínica odontológica de estética avançada situada em Curitiba, dotada de laboratório digital próprio.

Viabilidade em princípio, porém, não é viabilidade demonstrada. A adoção de IA por organizações fora do eixo das grandes empresas de tecnologia esbarra em três gargalos recorrentes. O primeiro é a inflexibilidade dos chatbots monolíticos, que falham diante de tarefas que exigem especialização distinta em cada domínio. O segundo é o acoplamento entre inferência e persistência: sistemas que processam a resposta da IA e gravam o registro da interação no mesmo ciclo síncrono acumulam latência e ficam expostos à perda de dados. O terceiro, e mais crítico para o argumento aqui desenvolvido, é a opacidade da caixa-preta: sem um registro estruturado do comportamento do sistema, torna-se inviável auditar o raciocínio dos agentes e mensurar, de forma objetiva, custo, desempenho e qualidade das respostas. Coloca-se, assim, a pergunta que organiza este trabalho: é possível uma arquitetura que enfrente

os três gargalos simultaneamente, sustentando um sistema de IA conversacional especializado, auditável e economicamente compatível com o orçamento de uma organização de pequeno porte?

A resposta proposta constitui a tese deste trabalho: uma arquitetura serverless orientada a eventos é capaz de sustentar um sistema multiagente de IA com rastreabilidade total das interações, permitindo não apenas a automação de tarefas especializadas, mas também a mensuração objetiva de custo, desempenho e qualidade das respostas, condição para viabilizar a adoção de IA conversacional em pequenas e médias organizações. Essa tese afirma três proposições que precisam sustentar-se em conjunto: que o sistema funciona, atendendo demandas reais e heterogêneas; que é observável, tornando cada interação auditável via SQL sem acesso ao código-fonte; e que é economicamente viável, com custo mensurável e proporcional ao uso. As três são interdependentes: a funcionalidade sem observabilidade seria uma caixa-preta, a observabilidade sem viabilidade ficaria restrita a orçamentos de pesquisa e desenvolvimento, e a viabilidade sem funcionalidade demonstrada seria promessa sem lastro empírico.

Desse enunciado deriva o objetivo geral do trabalho: projetar, implantar e avaliar empiricamente, em uma clínica odontológica de pequeno porte, uma arquitetura serverless orientada a eventos para um sistema multiagente de IA conversacional, capaz de automatizar tarefas especializadas com rastreabilidade total e custo mensurável. Esse objetivo desdobra-se em quatro objetivos específicos:

- a) implementar o ecossistema multiagente, Orquestrador e oito agentes especialistas, dotado de roteamento multimodal, memória de sessão e loop de auto-avaliação;
- b) construir o pipeline de dados orientado a eventos ($S3 \rightarrow \text{Lambda} \rightarrow \text{Athena}$) que persiste e normaliza cada interação em um data warehouse consultável;
- c) demonstrar a rastreabilidade total do comportamento do sistema a partir exclusivamente dos dados persistidos;
- d) mensurar objetivamente o custo, o desempenho e a qualidade das respostas geradas.

A justificativa do trabalho apoia-se em uma lacuna apontada pela literatura. Khanagar et al. (2021), ao revisarem a produção sobre IA na odontologia, encontraram concentração

quase integral em diagnóstico por imagem , detecção de cáries, fraturas radiculares e análise cefalométrica ,, sem sistemas conversacionais, automação administrativa ou geração de documentação jurídica no domínio. O sistema aqui apresentado ocupa parte desse espaço ao cobrir, em operação real, domínios clínicos, comerciais, laboratoriais e jurídicos de forma integrada. A contribuição principal, contudo, é menos sobre a odontologia e mais sobre um método: tratar a observabilidade não como recurso de depuração, mas como camada de mensuração de baixo custo que torna o comportamento de um sistema de IA auditável via SQL , deslocando a barreira de adoção de IA em pequenas e médias organizações do custo de infraestrutura para o conhecimento técnico de implantação.

O restante deste trabalho organiza-se em quatro capítulos. O Capítulo 2 apresenta o referencial teórico, percorrendo os fundamentos dos sistemas multiagentes, dos modelos de linguagem, das arquiteturas orientadas a eventos e dos pipelines analíticos serverless. O Capítulo 3 detalha a metodologia, descrevendo a arquitetura em cinco camadas, o funcionamento dos agentes e o fluxo de dados que sustenta a análise. O Capítulo 4 apresenta os resultados da implantação, organizados em torno das três proposições da tese. O Capítulo 5 discute as limitações identificadas, sintetiza as conclusões e aponta os trabalhos futuros.

2 REFERENCIAL TEÓRICO

2.1 Modelos de Linguagem de Grande Escala (LLMs)

2.1.1 Arquitetura Transformer e o Mecanismo de Atenção

Até meados da década de 2010, a abordagem dominante para processamento de sequências textuais eram as redes neurais recorrentes (RNNs) e suas variantes LSTM, que processavam tokens de forma serial, impondo duas limitações estruturais: impossibilidade de paralelizar o treinamento de forma eficiente e dificuldade de preservar relações entre elementos distantes — fenômeno conhecido como vanishing gradient.

Em 2017, Vaswani et al. propuseram a arquitetura Transformer, baseada exclusivamente em self-attention, sem componente recorrente. O trabalho "Attention Is All You Need" demonstrou ser possível superar o estado da arte em tradução automática com uma arquitetura inteiramente paralela e computacionalmente mais eficiente (VASWANI et al., 2017). O mecanismo opera calculando, para cada token, pontuações que determinam o quanto a representação dos demais tokens deve influenciá-lo — processo executado em múltiplas "cabeças" simultâneas (multi-head attention), cada uma capturando diferentes relações semânticas e sintáticas. Ao contrário das RNNs, a self-attention associa diretamente quaisquer dois tokens da sequência, independentemente de suas posições relativas, o que é essencial para dependências de longa distância — como associar "cardíaco" e "anticoagulante" a uma conduta terapêutica em contexto clínico. Os LLMs modernos — incluindo as famílias Gemini, GPT e Qwen utilizadas neste trabalho — são construídos sobre variantes aprofundadas dessa arquitetura, com centenas de camadas e bilhões de parâmetros (BROWN et al., 2020).

2.1.2 Do Pré-treinamento ao Ajuste Fino

Um LLM passa por um ciclo de treinamento em duas fases, cuja compreensão justifica a decisão de especializar agentes exclusivamente por meio de instruções textuais, sem retreinamento.

Pré-treinamento. O modelo é exposto a um corpus massivo — livros, artigos científicos, páginas da web e documentos de domínios variados — e treinado para prever o próximo token dado o contexto anterior. Esse objetivo força a internalização de estruturas gramaticais, fatos sobre o mundo e padrões de raciocínio (BROWN et al., 2020). O resultado é o que Bommasani et al.

(2021) denominam *foundation model*: um sistema de propósito geral que serve de base para aplicações especializadas sem modificação arquitetural.

Ajuste fino com feedback humano (RLHF). O modelo pré-treinado é um preditor estatístico não alinhado a objetivos humanos específicos. Ouyang et al. (2022) sistematizaram o RLHF (Reinforcement Learning from Human Feedback): coletam-se demonstrações de comportamento desejado, treina-se um modelo de recompensa a partir de avaliações humanas e otimiza-se o modelo de linguagem guiado por esse modelo de recompensa. O resultado é um sistema cujas respostas são significativamente mais úteis e alinhadas às intenções do usuário (OUYANG et al., 2022).

Implicação arquitetural. O entendimento desse ciclo justifica a especialização dos agentes via *system prompts* — instruções textuais fornecidas como contexto inicial —, sem nenhum retreinamento. Os *foundation models* já internalizaram, nas fases de pré-treinamento e RLHF, o conhecimento de domínio necessário para operar como especialistas em odontologia, atendimento ao cliente ou geração de documentos jurídicos. O *system prompt* não ensina o modelo — ele restringe e orienta um modelo que já sabe, tornando a arquitetura economicamente acessível às organizações de pequeno e médio porte que constituem o público-alvo do sistema.

2.1.3 LLMs Multimodais: Integração de Texto e Imagem

A extensão do paradigma Transformer para o processamento conjunto de texto e imagens — denominada multimodalidade — tem impacto direto sobre três agentes do sistema: o Agente A (Leitor de Imagens), o Agente B (Especialista Clínico) e o Agente C (Visão Técnica).

A estratégia dominante consiste em associar ao Transformer de linguagem um encoder visual que traduz o conteúdo de uma imagem em representações vetoriais processáveis conjuntamente com os tokens textuais. Um dos encoders mais amplamente adotados é o CLIP (Contrastive Language–Image Pretraining), de Radford et al. (2021), treinado para alinhar representações de imagens e descrições textuais em um mesmo espaço vetorial. O GPT-4 com capacidades visuais (OPENAI, 2023) é um exemplo dessa classe que demonstrou desempenho expressivo em raciocínio visual complexo, incluindo análise de imagens médicas.

No sistema proposto, o Agente B recebe radiografias e fotografias intraorais para identificar achados anatômicos; o Agente C processa imagens de rótulos de insumos e erros de equipamentos; e o Agente A opera como pré-processador universal, classificando o tipo e conteúdo de cada imagem recebida em Base64 e devolvendo ao Orquestrador a informação necessária para o roteamento ao especialista adequado. A existência desses três agentes é uma consequência direta do domínio odontológico, no qual parcela significativa das informações relevantes circula em formato de imagem.

2.1.4 Engenharia de Prompts e Chain-of-Thought

A especialização dos agentes é obtida por prompt engineering: a estruturação cuidadosa de instruções textuais que condicionam o comportamento do modelo para uma tarefa ou domínio específico, sem modificação de parâmetros.

Modalidades de prompting. Brown et al. (2020) sistematizaram a distinção entre zero-shot — apenas a descrição da tarefa, sem exemplos — e few-shot — instrução acrescida de exemplos demonstrativos, que melhora substancialmente o desempenho em tarefas complexas. Os system prompts dos agentes deste sistema operam em regime zero-shot: as instruções definem perfil, capacidades autorizadas, limites absolutos, tom e contrato de saída JSON, sem exemplos extensos — escolha justificada pela natureza dinâmica das entradas (texto, imagem, áudio), que torna a construção de exemplos representativos impraticável.

Chain-of-Thought (CoT). Wei et al. (2022) demonstraram que induzir o modelo a externalizar seu raciocínio passo a passo antes de produzir a resposta final melhora consistentemente o desempenho em tarefas de raciocínio aritmético, lógico e de senso comum. No sistema proposto, o CoT se manifesta em dois componentes: o Orquestrador estrutura um campo "plano" em seu JSON de saída — uma lista sequencial de agentes a acionar —, constituindo uma forma de CoT aplicada ao planejamento; o Agente H percorre um checklist explícito de critérios de qualidade antes de emitir seu veredito, impondo raciocínio auditável sobre a resposta gerada, padrão formalizado por Shinn et al. (2023) no framework Reflexion.

Estrutura dos system prompts. Cada prompt define: (1) o papel do agente; (2) as capacidades autorizadas; (3) os limites absolutos; (4) o tom e registro linguístico; e (5) o contrato de saída em schema JSON. Essa estrutura reduz o espaço de respostas plausíveis do foundation model a um subconjunto compatível com os requisitos operacionais, éticos e legais do domínio

odontológico — a instrumentalização do conhecimento generalista do LLM para um contexto especializado, exatamente o paradigma descrito por Bommasani et al. (2021).

2.2 Sistemas Multi-Agentes de IA (MAS)

A seção anterior estabeleceu os LLMs como motores individuais de raciocínio. Contudo, um único modelo respondendo simultaneamente a dúvidas clínicas, comerciais, laboratoriais e jurídicas produziria comportamento inconsistente e rastreabilidade reduzida. A solução adotada neste trabalho é a decomposição dessas responsabilidades em agentes autônomos e especializados, coordenados por um componente central — padrão que a literatura classifica como Sistema Multi-Agentes (Multi-Agent System — MAS). As subseções a seguir estabelecem a base conceitual que fundamenta essa escolha.

2.2.1 Definição e Propriedades de Agentes Inteligentes

Wooldridge e Jennings (1995) definem um agente como um sistema computacional situado em um ambiente, capaz de agir de forma autônoma com vistas a satisfazer seus objetivos de design. Essa definição distingue o agente de uma função passiva: ele percebe o ambiente, delibera e age com intenção. Os autores formalizam quatro propriedades que caracterizam um agente inteligente:

Autonomia: o agente opera sem intervenção direta de humanos ou outros componentes, exercendo controle sobre seu próprio estado interno. No sistema proposto, o Agente B decide de forma autônoma quais achados clínicos são relevantes em uma radiografia e se o campo `flag_risco` deve ser ativado — sem que o Orquestrador intervenha nesse raciocínio.

Reatividade: o agente percebe seu ambiente e responde a mudanças nele de forma oportuna. O ambiente de cada agente é o JSON recebido do Orquestrador — texto, perfil do remetente e eventual imagem. O Agente D produz respostas estruturalmente distintas diante de um paciente que objetou preço versus um que adiou por razões emocionais, porque reage ao conteúdo específico do contexto.

Pró-atividade: o agente toma iniciativa orientada a objetivos, sem esperar instrução explícita para cada etapa. O Agente H materializa essa propriedade ao percorrer proativamente um checklist de critérios de qualidade — clareza, precisão técnica, detecção de alucinação — sem que nenhum componente externo defina quais dimensões avaliar.

Sociabilidade: o agente interage com outros agentes por meio de contratos de comunicação explícitos. No sistema proposto, esse contrato é o protocolo JSON compartilhado entre Orquestrador e especialistas: cada mensagem carrega campos de intenção, instrução e resultado, viabilizando coordenação sem acoplamento direto.

Essa quadruplicidade conceitual demarca a linha entre o sistema proposto e o chatbot monolítico que ele substitui: um sistema monolítico não é autônomo (suas respostas derivam de um único LLM sem especialização), não é pró-ativo e não é social. A adoção do MAS resolve diretamente os três gargalos enunciados na motivação: inflexibilidade, acoplamento e opacidade.

2.2.2 Taxonomias de Arquiteturas Multi-Agentes

A literatura converge para duas grandes famílias arquiteturais de MAS. Na arquitetura plana (peer-to-peer), todos os agentes operam no mesmo nível hierárquico e a coordenação emerge de protocolos de negociação direta — leilão de tarefas, votação ou broadcast. Esse modelo é adequado quando a especialização dos agentes é homogênea e as tarefas são divisíveis em unidades independentes de complexidade similar (WOOLDRIDGE; JENNINGS, 1995).

Na arquitetura hierárquica, um agente de controle coordena agentes subordinados, delegando subtarefas, consolidando resultados e mantendo o estado global da interação. Esse modelo é recomendado quando o sistema enfrenta três condições simultâneas: as tarefas são heterogêneas e exigem roteamento inteligente; a especialização dos agentes é assimétrica; e a rastreabilidade das decisões é um requisito (WANG et al., 2024). Nenhum especialista precisa conhecer os demais — apenas o coordenador mantém a visão global, reduzindo acoplamento e tornando o comportamento do sistema auditável de forma centralizada.

A escolha hierárquica neste trabalho decorre diretamente dessas condições: as mensagens variam de dúvidas clínicas sobre interações medicamentosas a erros de impressão 3D e solicitações de termos de consentimento. O requisito de rastreabilidade total — uma das três afirmações centrais da tese — exige um ponto de controle único e auditável, o que a arquitetura plana não garante estruturalmente.

2.2.3 O Padrão Orquestrador-Trabalhador

Dentro das arquiteturas hierárquicas, o padrão Orquestrador-Trabalhador (Orchestrator-Workers) é a instância mais diretamente aplicável ao sistema proposto. Um agente central — o Orquestrador — recebe a tarefa completa, decompõe a intenção do usuário em subtarefas e delega cada uma ao agente especialista (worker) mais adequado. Os workers não se comunicam entre si; toda coordenação passa pelo Orquestrador, que é o único componente com acesso ao estado global da interação.

Uma propriedade crítica desse padrão é a separação entre raciocínio de coordenação e execução especializada: o Orquestrador sabe quando acionar o especialista clínico, mas não precisa saber como analisar uma radiografia. Essa separação viabiliza a adição de novos agentes sem modificar a lógica do Orquestrador — basta registrar o novo worker com uma descrição de quando deve ser acionado.

Kumar (2024) descreve a implementação desse padrão no n8n: o Orquestrador é configurado como um nó agent, que expõe um loop de raciocínio completo; cada especialista é um nó agentTool — uma ferramenta nomeada cujo acionamento emerge do raciocínio do próprio modelo, não de regras codificadas manualmente. O campo plano do JSON de saída do Orquestrador registra, por interação, quais agentes foram ativados e com quais instruções — tornando o fluxo decisório completamente rastreável via SQL no Amazon Athena (Seção 3.7).

2.2.4 Frameworks de Raciocínio e Ação: ReAct, Reflexion, AutoGen e LangGraph

ReAct. Yao et al. (2023) propuseram o framework ReAct (Reasoning + Acting), no qual o agente alterna entre blocos de Thought (raciocínio interno), Action (chamada a uma ferramenta) e Observation (resultado incorporado ao contexto), em um ciclo iterativo. Esse loop permite planejamento adaptativo e produz um rastro de raciocínio auditável. O nó agent do n8n executa implicitamente esse padrão ao deliberar sobre a sequência de agentes a acionar.

Reflexion. Shinn et al. (2023) introduziram um mecanismo de auto-avaliação verbal: após gerar uma resposta, o agente avalia sua própria saída e refina iterativamente sem retreinamento. O Agente H deste sistema implementa esse padrão — ele recebe a resposta de qualquer worker, avalia critérios de qualidade estruturados e, quando detecta necessidade de revisão, força o Orquestrador a regenerar a resposta antes de devolvê-la ao usuário (Seção 3.4.5).

AutoGen e LangGraph. Wu et al. (2023) apresentaram o AutoGen como framework de conversação multi-agente configurável, onde qualquer topologia de MAS pode ser expressa como um grafo de trocas de mensagens. O LangGraph (2024) representa o estado da arte em controle de estado explícito: agentes são nós de um grafo dirigido com transições tipadas e persistência de estado entre execuções, oferecendo garantias formais de fluxo.

Posicionamento do n8n. O n8n não é um framework de agentes em sentido estrito, mas uma plataforma de automação com suporte nativo a nós LangChain que viabiliza o padrão Orquestrador-Trabalhador sem código. Conforme Kumar (2024), isso reduz drasticamente a barreira de entrada para organizações sem equipe de engenharia dedicada. O custo é menor controle formal de estado e paralelismo mais limitado em comparação ao LangGraph — trade-off tratado como limitação reconhecida na Seção 3.8.3.

2.2.5 Memória e Gestão de Contexto em Agentes LLM

Uma propriedade estrutural dos LLMs com implicação direta sobre o design de agentes conversacionais é a ausência de memória nativa entre requisições: cada chamada à API é stateless — o modelo não retém informação sobre interações anteriores sem que o histórico seja explicitamente fornecido como parte do contexto (VASWANI et al., 2017). A gestão do contexto entre turnos é responsabilidade da camada de aplicação.

Zhang et al. (2024) sistematizam três estratégias complementares de memória para agentes LLM:

Memória de buffer (in-context buffer): os N turnos mais recentes são inseridos diretamente na janela de contexto da requisição seguinte. É a estratégia mais simples — sem infraestrutura externa — mas seu custo de inferência é proporcional ao histórico acumulado, e conversas longas podem degradar o foco do modelo em conteúdo recente.

Memória externa (external memory): o histórico é persistido em armazenamento externo — banco de dados relacional, Redis ou chave-valor — e recuperado sob demanda antes de cada requisição. Desacopla a capacidade de memória da janela de contexto ao custo de latência adicional na leitura.

Memória semântica (semantic memory): o histórico é convertido em embeddings e recuperado por similaridade com a consulta atual, não por posição temporal. Adequada a históricos longos e heterogêneos, nos quais a relevância de um turno passado é temática, não cronológica.

O Orquestrador deste sistema adota memória externa com buffer de tamanho fixo: o histórico de cada sessão é persistido no Redis com janela de 15 turnos e TTL de 3.600 segundos. A janela de 15 turnos cobre a maioria das sessões de trabalho da clínica sem acumular contexto irrelevante; o TTL garante que sessões abandonadas não consumam memória indefinidamente. A preferência por recência temporal sobre recuperação semântica é justificada pelo perfil de uso: sessões curtas e tópicas, nas quais a recência é critério de relevância suficiente — e a ausência de um pipeline de embeddings reduz dependência de infraestrutura adicional, alinhada ao princípio de viabilidade econômica para PMEs que orienta o design do sistema (Seção 3.1.3).

2.3 Princípios Arquiteturais de Software para Sistemas de IA

As seções anteriores estabeleceram os fundamentos dos componentes inteligentes do sistema: LLMs como motores de raciocínio e sistemas multi-agentes como estrutura de coordenação. Contudo, a capacidade de um MAS operar de forma confiável e economicamente mensurável em produção depende não apenas de como os agentes raciocinam, mas de como a infraestrutura ao redor deles é arquitetada. Três princípios de engenharia de software estruturam as decisões arquiteturais deste trabalho: separação de responsabilidades, arquitetura orientada a eventos e computação serverless.

2.3.1 Separação de Responsabilidades (SoC)

A separação de responsabilidades (Separation of Concerns — SoC) é um princípio enunciado por Dijkstra (1974), segundo o qual sistemas complexos devem ser decompostos em partes que tratem de aspectos distintos do problema, de modo que cada parte concentre sua atenção em uma única preocupação bem definida e se comunique com as demais exclusivamente por meio de contratos explícitos. A motivação central do princípio é cognitiva: a capacidade de lidar com complexidade depende da habilidade de focar em um aspecto de cada vez, sem que esse foco exija a consideração simultânea de todos os demais (DIJKSTRA, 1974).

Na engenharia de software, a SoC se materializa na arquitetura em camadas: cada camada oferece serviços à camada superior e consome serviços da camada inferior, ignorando os detalhes de implementação de todas as demais. Essa estrutura garante que a modificação interna

de uma camada — desde que o contrato de interface seja preservado — não force alterações em cascata nas demais, propriedade que Kleppmann (2017) denomina independência de deployment. Em sistemas de processamento de dados em múltiplos estágios, essa independência é especialmente relevante: a lógica de transformação pode evoluir sem alterar a camada de ingestão, e a camada analítica pode ser substituída sem que o produtor de dados precise ser modificado.

Uma consequência direta da SoC é a contenção de falhas. Em sistemas nos quais responsabilidades distintas estão acopladas em um mesmo componente, uma falha em qualquer função pode comprometer o comportamento global. O isolamento entre subsistemas é condição necessária para que cada um seja testado, monitorado e recuperado de forma independente (KLEPPMANN, 2017). No contexto de sistemas baseados em LLMs, Wooldridge e Jennings (1995) reforçam essa dimensão: agentes inteligentes devem operar sobre domínios restritos, sem acesso ao estado interno dos demais componentes — o que torna a SoC não apenas uma diretriz de engenharia, mas uma exigência da própria definição de agente autônomo.

2.3.2 Arquitetura Orientada a Eventos (EDA)

A Arquitetura Orientada a Eventos (Event-Driven Architecture — EDA) é um paradigma no qual o fluxo de execução é determinado pela ocorrência e propagação de eventos. Michelson (2006) define evento como um fato notável que ocorre dentro ou fora de um sistema e que é imediatamente disseminado a todas as partes interessadas para avaliação e eventual ação. A autora destaca a propriedade fundamental do modelo: o produtor do evento conhece apenas o fato de que o evento ocorreu, sem qualquer conhecimento sobre quem o consumirá ou de que forma (MICHELSON, 2006).

A EDA se organiza em torno de três papéis distintos. O produtor detecta a ocorrência de um fato e o publica sem aguardar confirmação de consumo. O canal é o mecanismo de transporte — um barramento de mensagens, uma fila ou, em arquiteturas baseadas em armazenamento de objetos, o próprio evento de escrita em um bucket. O consumidor subscreve ao canal e reage ao evento de forma assíncrona, sem acoplamento direto ao produtor (MICHELSON, 2006).

Esse modelo contrasta com arquiteturas síncronas tradicionais, nas quais um componente invoca explicitamente outro e aguarda resposta. Em cadeias síncronas, a latência total é a soma das latências individuais, e uma falha em qualquer elo interrompe toda a execução. Na EDA, o

produtor conclui sua operação imediatamente após publicar o evento, e o processamento subsequente ocorre em paralelo — o que Kleppmann (2017) denomina desacoplamento temporal. As consequências práticas são três: resiliência (falha no consumidor não afeta o produtor, e o evento permanece disponível para reprocessamento), escalabilidade horizontal (cada evento gera uma unidade de trabalho processável em paralelo, sem coordenação explícita do produtor) e extensibilidade (novos consumidores podem ser adicionados sem modificar o produtor, pois o contrato é o evento em si).

2.3.3 Computação Serverless

O modelo serverless designa a classe de serviços em nuvem na qual o provedor gerencia integralmente a infraestrutura de execução — provisionamento, escalonamento e disponibilidade —, cabendo ao desenvolvedor apenas definir o código e as condições que o disparam. Jonas et al. (2019) distinguem dois componentes complementares desse modelo: Functions as a Service (FaaS), que abstrai a execução de funções sob demanda — como o AWS Lambda —, e Backend as a Service (BaaS), que abstrai serviços de armazenamento e consulta sem servidor fixo — como o Amazon S3 e o Amazon Athena. Os autores argumentam que a combinação de FaaS e BaaS constitui o paradigma serverless completo, no qual nenhuma camada do sistema requer infraestrutura provisionada continuamente (JONAS et al., 2019).

Wang et al. (2018), em estudo empírico sobre plataformas serverless, identificam que esse modelo apresenta elasticidade estruturalmente distinta de instâncias dedicadas: o escalonamento é automático e por invocação, e o custo de capacidade ociosa é eliminado porque não há infraestrutura mantida em períodos sem demanda. Os autores observam que, em cargas de trabalho irregulares — com picos intercalados por períodos de ociosidade —, o modelo serverless apresenta vantagem econômica estrutural, convertendo custo fixo em custo estritamente variável e proporcional ao uso real (WANG et al., 2018).

Sob a perspectiva analítica, Armbrust et al. (2021) demonstram que a combinação de armazenamento de objetos — S3 — com motores de consulta desacoplados — Athena — concretiza o padrão lakehouse: unificação das capacidades de um data lake (armazenamento de dados brutos em escala) com as de um data warehouse (consulta analítica estruturada via SQL), sobre infraestrutura inteiramente serverless. Os autores apontam que esse modelo é adequado a cargas analíticas orientadas a eventos, nas quais a demanda por consulta é irregular e o volume

de dados cresce proporcionalmente ao uso do sistema (ARMBRUST et al., 2021). A diferença econômica frente a bancos de dados relacionais dedicados é estrutural: uma instância gerenciada como Amazon RDS exige pagamento contínuo por hora, independentemente do volume de transações; o modelo S3 + Athena cobra exclusivamente pelo volume escaneado por consulta e pelo armazenamento efetivo — tornando o custo do sistema diretamente mensurável a partir do uso real.

2.4 Pipelines de Dados: Do Data Lake ao Lakehouse Analítico

2.4.1 Data Lake: Conceito e Desafios

O termo data lake descreve um repositório centralizado capaz de armazenar dados em formato bruto — tal como foram gerados na origem —, independentemente de estrutura, modalidade ou finalidade analítica futura. Nargesian et al. (2019) definem data lake como um sistema que ingere dados de múltiplas fontes heterogêneas sem exigir transformação prévia, adiando a definição do esquema para o momento da consulta — abordagem denominada schema on read, em contraposição ao schema on write dos Data Warehouses tradicionais. Essa distinção é estruturalmente relevante: um Data Warehouse convencional exige que colunas, tipos e restrições sejam declarados antes do primeiro registro, conferindo consistência a custo de rigidez que se torna onerosa quando os dados de origem variam em estrutura — como ocorre quando novos agentes são adicionados ao sistema e os campos do JSON de saída se expandem.

A flexibilidade do Data Lake, no entanto, introduz desafios próprios sistematizados por Nargesian et al. (2019) em três dimensões. A governança é comprometida quando arquivos mal formatados, duplicados ou de fontes não autorizadas se acumulam sem detecção. O data swamp ocorre quando a ausência de metadados organizacionais torna o repositório inutilizável: sem saber o que cada arquivo contém, quem o gerou ou a que período se refere, nenhuma consulta analítica pode ser construída de forma confiável. A rastreabilidade exige que o nome e a localização dos arquivos carreguem informação suficiente para identificação direta, sem necessidade de varrer o conteúdo do repositório.

No sistema proposto, essas três dimensões são endereçadas na organização do blavur-data-lake. A governança é garantida pelo acesso restrito ao bucket via token de autenticação estático — única fonte de escrita é o workflow de persistência do n8n. O risco de data swamp é mitigado pela convenção de nomenclatura {timestamp}-{source}-{sessionId}-{user}.json, que torna

canal, sessão, remetente e momento da gravação legíveis diretamente na listagem do bucket. A rastreabilidade é complementada pela organização em pastas diárias no formato DD-MM-YYYY. Essas escolhas de design materializam, no nível da infraestrutura, as respostas concretas aos desafios que Nargesian et al. (2019) identificam como recorrentes em implementações sem governo explícito.

2.4.2 ETL Orientado a Eventos

O processo de transformação de dados brutos em registros analíticos estruturados é denominado ETL — sigla que descreve três etapas: Extract (extração da fonte de origem), Transform (normalização e reestruturação para um formato analítico) e Load (carregamento no destino consultável). No modelo batch — historicamente dominante —, o ETL é agendado para execução periódica em janelas fixas: todos os arquivos acumulados desde a última execução são processados em lote. Esse modelo é simples, mas introduz atraso estrutural entre a geração do dado e sua disponibilidade analítica, podendo mascarar falhas de processamento por horas (KLEPPMANN, 2017).

No modelo orientado a eventos, o ETL é disparado imediatamente a cada novo dado na fonte, sem agendamento. Kleppmann (2017) argumenta que esses sistemas devem armazenar o evento original de forma imutável antes de qualquer transformação: se a transformação falhar, o dado bruto permanece disponível para reproprocessamento sem exigir a reexecução do processo que o gerou. O autor alerta, porém, para o risco de falhas silenciosas: ao contrário do batch — que produz um log consolidado ao término de cada janela —, uma falha em evento individual pode passar despercebida na ausência de mecanismos de monitoramento explícitos, como filas de mensagens mortas (dead-letter queues).

No sistema proposto, a AWS Lambda implementa o ETL orientado a eventos: cada arquivo .json depositado no `blanvur-data-lake` dispara automaticamente uma invocação independente, sem agendamento. O dado bruto permanece imutável no bucket de origem, garantindo a resiliência descrita por Kleppmann (2017). A ausência de uma dead-letter queue nesta implementação é reconhecida como limitação da arquitetura e discutida na Seção 3.8.3.

2.4.3 O Padrão Lakehouse e a Consulta Analítica com S3 + Athena

Armbrust et al. (2021) formalizaram o padrão Lakehouse como a abordagem que combina o armazenamento flexível de um data lake com a capacidade de consulta analítica de um data warehouse, sobre infraestrutura inteiramente serverless e sem cluster dedicado. No modelo Lakehouse, os dados residem em armazenamento de objetos — como o Amazon S3 — em formatos abertos como JSON, e são consultados por motores desacoplados que os leem diretamente, sem ingestão prévia. O Amazon Athena é o representante desse modelo no ecossistema AWS: executa SQL sobre arquivos no S3 e cobra exclusivamente pelo volume de dados escaneado por consulta (ARMBRUST et al., 2021).

A adequação técnica dessa escolha ao sistema proposto decorre da distinção entre dois regimes de operação. Cargas OLTP (Online Transaction Processing) são caracterizadas por transações frequentes e curtas com exigência de consistência imediata. Cargas OLAP (Online Analytical Processing) são caracterizadas por consultas ad-hoc orientadas à agregação de volumes históricos, sem exigência de consistência imediata. A camada analítica deste sistema opera exclusivamente em regime OLAP: as consultas são retrospectivas, agregativas e irregulares — executadas ao fim do período de implantação para produzir métricas, não para suportar transações em tempo real. Adotar um banco relacional dedicado como o PostgreSQL implicaria pagamento contínuo por capacidade provisionada independentemente do volume de consultas; o modelo S3 + Athena converte esse custo fixo em custo estritamente variável, tornando a viabilidade econômica do sistema diretamente mensurável (ARMBRUST et al., 2021).

2.4.4 Particionamento Hive-Style

A adoção do padrão Lakehouse elimina a necessidade de cluster dedicado, mas introduz um desafio de desempenho: em armazenamento de objetos sem organização estruturada, qualquer consulta filtrada por critério temporal precisaria varrer todos os arquivos do repositório para localizar os registros relevantes — tornando custo e latência proporcionais ao volume histórico acumulado. O particionamento Hive-style resolve esse problema.

Kleppmann (2017) descreve o particionamento por intervalo de chave (key-range partitioning) como a estratégia na qual os dados são organizados de modo que cada partição concentre um subconjunto contíguo de valores de uma chave de interesse — como uma faixa de datas. O motor de consulta, ao receber um predicado de filtragem sobre essa chave, determina com precisão quais partições são relevantes e quais podem ser ignoradas sem varrimento. Armbrust

et al. (2021) aplicam esse princípio ao padrão Lakehouse sob o nome data skipping via partitioning: ao organizar os dados em prefixos chave=valor — como year=AAAA/month=MM/day=DD/ —, o motor executa partition pruning, limitando o scan exclusivamente às partições que satisfazem o predicado da consulta.

No sistema proposto, esse padrão é implementado pela função `build_partition_prefix` da AWS Lambda, que calcula o prefixo a partir do campo `time_msg_start` do JSON bruto, registrado em UTC — decisão que garante consistência das partições independentemente do fuso horário do canal ou do servidor. O padrão é aplicado uniformemente às três tabelas de destino — `messages`, `agents_plan` e `agents_actions` —, viabilizando junções temporais sem ambiguidade entre tabelas. Conforme apontado por Wang et al. (2018), o modelo serverless converte custos fixos em custos variáveis; o particionamento Hive-style é o mecanismo análogo na camada de armazenamento, garantindo que o custo de cada consulta analítica seja proporcional ao dado efetivamente necessário para respondê-la — e não ao volume histórico total do sistema.

2.5 IA Aplicada ao Setor de Saúde e Odontologia

2.5.1 Transformação Digital em Organizações de Pequeno e Médio Porte

A adoção de IA por pequenas e médias empresas (PMEs) é historicamente limitada por três barreiras estruturais: o custo de infraestrutura, a ausência de equipes técnicas internas e a inadequação de soluções comerciais dimensionadas para grandes organizações. Bommasani et al. (2021) descrevem a emergência dos foundation models como uma ruptura nessa lógica: ao encapsular conhecimento de propósito geral durante o pré-treinamento, esses modelos podem ser adaptados para domínios específicos por meio de instruções textuais simples, sem retreinamento ou infraestrutura própria. Uma clínica odontológica pode, portanto, operar agentes especialistas fornecendo apenas um system prompt ao modelo — eliminando a principal barreira técnica de acesso à IA (BOMMASANI et al., 2021). A barreira de infraestrutura, por sua vez, é endereçada pelo modelo serverless: conforme Wang et al. (2018), ele converte o custo fixo de servidores dedicados em custo estritamente variável e proporcional ao uso, tornando a viabilidade econômica do sistema diretamente mensurável — e, portanto, um resultado quantitativo deste trabalho.

2.5.2 Chatbots e Assistentes Virtuais em Contextos Clínicos

A aplicação de LLMs em saúde é uma das linhas de pesquisa de maior crescimento recente. Lee et al. (2023) demonstraram que modelos de grande escala respondem perguntas médicas com desempenho comparável ao de residentes em benchmarks padronizados, documentando simultaneamente os limites éticos centrais dessa aplicação: sistemas de IA em saúde operam em regime de apoio à decisão — nunca de substituição do julgamento profissional. Esse princípio está diretamente refletido no Agente B deste trabalho, cujo system prompt reforça em toda resposta clínica que a decisão final cabe ao profissional habilitado, e no campo `flag_risco: true`, que aciona o Agente H como auditor obrigatório em respostas com potencial impacto sobre a segurança do paciente.

No domínio odontológico, Khanagar et al. (2021) revisaram 43 estudos sobre IA na odontologia publicados entre 2000 e 2020, concluindo que a literatura se concentra quase integralmente em diagnóstico por imagem — detecção de cáries, fraturas radiculares e análise cefalométrica — com uso predominante de CNNs sobre radiografias. Nenhum dos estudos analisados aborda sistemas conversacionais, automação administrativa ou geração de documentação jurídica no contexto odontológico. Essa lacuna é o espaço de contribuição original deste trabalho: uma clínica odontológica opera simultaneamente em domínios clínicos, comerciais, laboratoriais e jurídicos — multidisciplinaridade que a literatura existente não aborda de forma integrada, e que este sistema se propõe a cobrir com rastreabilidade total das interações.

2.6 Lei Geral de Proteção de Dados (LGPD)

A Lei nº 13.709/2018 (LGPD) estabelece o marco regulatório brasileiro para o tratamento de dados pessoais, definindo princípios como finalidade, necessidade, transparência e segurança que devem orientar qualquer operação de coleta, armazenamento ou processamento de dados de titulares identificáveis (BRASIL, 2018). O art. 11 da lei classifica dados de saúde como dados pessoais sensíveis, categoria sujeita a obrigações mais rigorosas — relevante para o presente sistema, cujas interações com pacientes odontológicos podem conter informações clínicas identificáveis.

Essa classificação tem implicações arquiteturais diretas: o princípio da segurança exige que os dados permaneçam sob controle do operador responsável e que seu fluxo seja rastreável, o que fundamenta a escolha pela hospedagem da camada de orquestração em infraestrutura própria, em detrimento de plataformas gerenciadas por terceiros que processariam os dados fora do controle do controlador.

3 METODOLOGIA

3.1 Visão Geral e Princípios Arquiteturais

O sistema proposto é estruturado em cinco camadas funcionais independentes: ingestão de mensagens, processamento inteligente via sistema multi-agente, persistência assíncrona em data lake, transformação e normalização via computação serverless, e consulta analítica. Essa organização decorre da aplicação deliberada de três princípios arquiteturais complementares — separação de responsabilidades, arquitetura orientada a eventos e execução serverless —, detalhados nas subseções a seguir.

3.1.1 Separação de Responsabilidades como Princípio Central

No sistema proposto, a SoC se manifesta estruturalmente: a camada de inteligência não contém lógica de persistência; a camada de persistência desconhece os agentes e o domínio odontológico; e a Lambda de ETL opera exclusivamente sobre o contrato de dados do arquivo JSON gravado no S3. Essa separação garante que uma falha na persistência não comprometa a resposta ao usuário — pois o retorno ocorre antes do disparo assíncrono da gravação — e que a substituição de um modelo de linguagem ou a adição de um agente especialista não exijam modificações no esquema analítico. Conforme apontado por Wooldridge e Jennings (1995), agentes inteligentes são caracterizados por autonomia e pró-atividade; a SoC preserva essa autonomia ao garantir que cada especialista opera sobre um domínio restrito, sem acesso direto ao estado interno dos demais.

3.1.2 Arquitetura Orientada a Eventos

No sistema proposto, esse padrão se manifesta na integração entre o data lake e a camada de ETL. O upload de qualquer arquivo *.json* no bucket S3 (*blanvur-data-lake*) constitui um evento que dispara automaticamente a AWS Lambda, sem nenhuma chamada explícita do n8n. O n8n não sabe que a Lambda existe; a Lambda não sabe como o n8n funciona. Ambos operam de forma autônoma, conectados apenas pelo evento de upload e pelo contrato de dados do arquivo. Em termos de resiliência, um erro no processamento da Lambda não afeta o dado de origem, que permanece intacto no S3 para reprocessamento. Em termos de escalabilidade, cada upload gera um evento independente, e a AWS gerencia o paralelismo sem configuração adicional.

3.1.3 Justificativa da Abordagem Serverless

A arquitetura proposta utiliza três serviços serverless da AWS de forma complementar: o Amazon S3, com cobrança por volume armazenado e operações realizadas; o AWS Lambda, com cobrança por invocação e tempo de execução — custo zero em períodos sem interações; e o Amazon Athena, com cobrança pelo volume de dados escaneado por consulta, sem cluster fixo. Essa combinação é especialmente adequada ao público-alvo: clínicas odontológicas de pequeno e médio porte apresentam uso concentrado em horários comerciais e ociosidade acentuada fora deles. Uma infraestrutura baseada em servidores dedicados implicaria pagar pelo pico de forma contínua; o modelo serverless converte esse custo fixo em custo estritamente variável, tornando a viabilidade econômica do sistema diretamente mensurável — e, portanto, um resultado quantitativo deste trabalho.

3.2 Visão em Camadas do Sistema

A arquitetura proposta neste trabalho é organizada em cinco camadas funcionais independentes, cada uma com responsabilidades bem delimitadas e interfaces de comunicação explícitas entre si. Essa organização decorre diretamente dos princípios arquiteturais apresentados na Seção 3.1 — separação de responsabilidades, arquitetura orientada a eventos e execução serverless — e tem como objetivo garantir que modificações em uma camada não impliquem alterações nas demais. A Figura 3.1 sintetiza, em nível conceitual, o fluxo de processo do sistema, do usuário até a camada de gestão, e serve como referência para as subseções que se seguem.

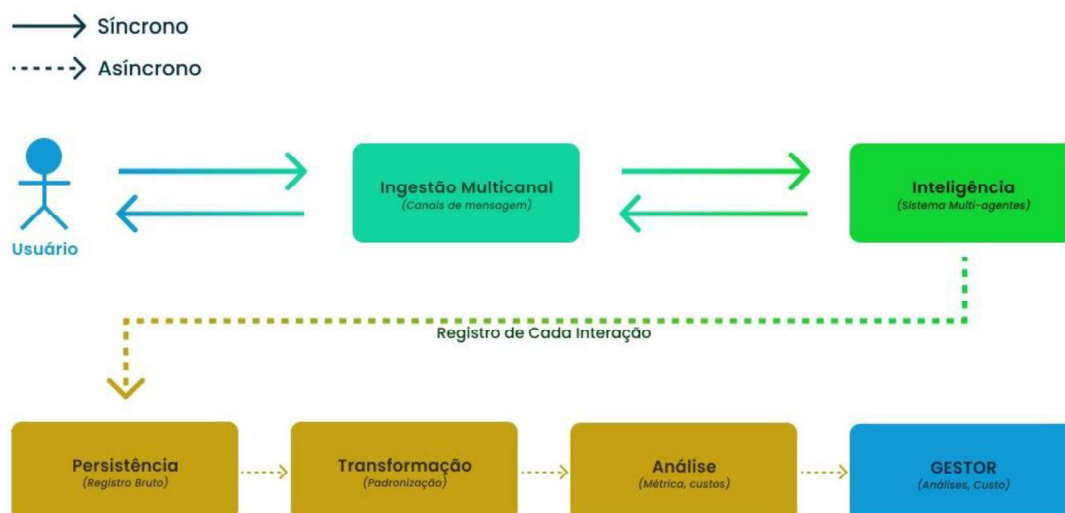


Figura 3.1 — Fluxo de processo do sistema, do usuário à camada de gestão. Setas sólidas indicam fluxo síncrono; setas tracejadas indicam fluxo assíncrono. Fonte: elaborado pelo autor.

A Figura 3.1 distingue dois fluxos de naturezas temporais diferentes. O primeiro é o ciclo síncrono de atendimento (setas sólidas): uma mensagem originada em qualquer canal é recebida pela camada de ingestão, encaminhada à camada de inteligência para processamento e devolvida ao usuário pelo mesmo caminho. É esse o único trajeto percebido pelo usuário e, portanto, o único que compõe o caminho crítico de latência. O segundo é o fluxo assíncrono de registro e análise (setas tracejadas): somente após a resposta ao usuário, cada interação é registrada e percorre, de forma desacoplada, as etapas de persistência, transformação e análise, até subsidiar as decisões do gestor com métricas operacionais e de custo. A distinção visual entre os dois fluxos é deliberada e materializa, em nível de processo, o princípio de separação de responsabilidades: o caminho de resposta ao usuário é independente do caminho de geração de dados analíticos, de modo que a observabilidade e a análise de custos do sistema não interferem na experiência de atendimento.

3.2.1 Diagrama Geral da Arquitetura

Enquanto a Figura 3.1 descreve o fluxo de processo em nível conceitual, a Figura 3.2 detalha a arquitetura técnica correspondente, explicitando os componentes e as tecnologias que implementam cada etapa.

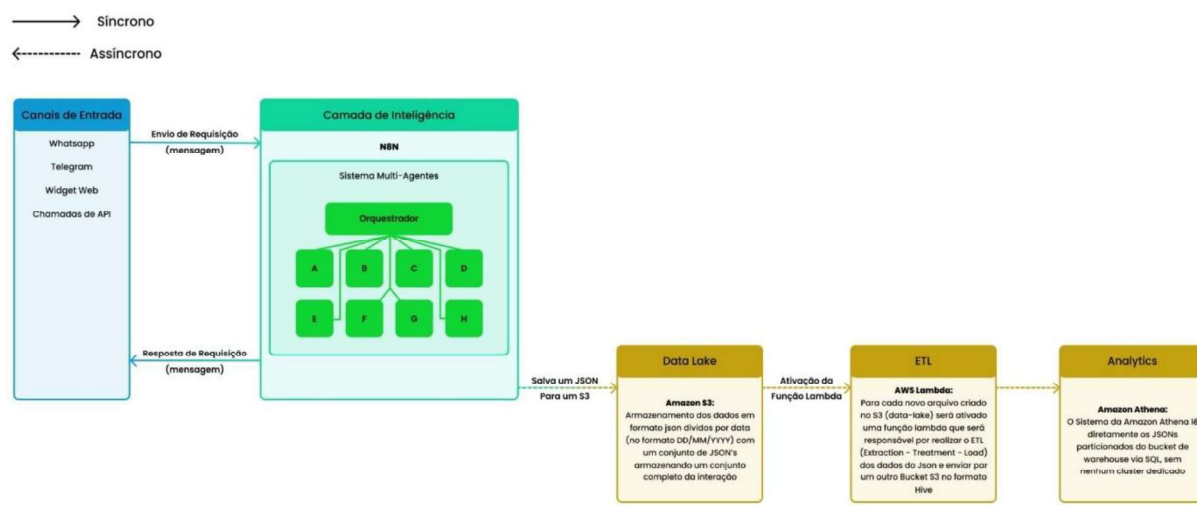


Figura 3.2 — Arquitetura técnica do sistema. Setas sólidas indicam fluxo síncrono; setas tracejadas indicam fluxo assíncrono. Fonte: elaborado pelo autor.

No caminho síncrono, as requisições provenientes dos canais de entrada — WhatsApp, Telegram, widget Web ou chamadas diretas via API — são enviadas à Camada de Inteligência, implementada no n8n, onde o Orquestrador classifica a intenção e delega o processamento aos agentes especialistas (A–H); a resposta retorna ao canal de origem pelo mesmo caminho. No caminho assíncrono, cada interação é gravada como um arquivo JSON no Data Lake (Amazon S3); esse upload constitui o evento que ativa automaticamente a função de ETL (AWS Lambda), responsável por normalizar os dados e gravá-los, em formato particionado, no bucket de Data Warehouse consultável via SQL pelo Amazon Athena. A correspondência entre as duas figuras é direta: cada bloco conceitual da Figura 3.1 encontra, na Figura 3.2, o componente tecnológico que o realiza.

3.2.2 Descrição das Cinco Camadas

As cinco camadas que compõem o pipeline são descritas a seguir em ordem de execução, da ingestão até a consulta analítica. O detalhamento técnico de cada uma é retomado nas subseções específicas do capítulo.

Camada de Ingestão: Responsável por receber requisições de qualquer canal de entrada — WhatsApp, Telegram, widget Web ou API — e normalizá-las em um objeto JSON estruturado antes de encaminhá-las à camada de inteligência. Implementada no n8n via webhook com autenticação por token, opera sem lógica de negócio ou persistência.

Camada de Inteligência (Sistema Multi-Agentes): Núcleo do sistema, responsável pela classificação de intenções, roteamento entre especialistas e geração de respostas. Implementada no n8n com nós LangChain, acessa os modelos de linguagem via OpenRouter e mantém memória de sessão via Redis com janela de 15 turnos e TTL de 3.600 segundos.

Camada de Persistência Bruta (Data Lake): Responsável por armazenar, de forma assíncrona e imutável, o registro completo de cada interação — resposta final, passos intermediários dos agentes e metadados de sessão. Implementada no Amazon S3, com arquivos JSON organizados em pastas diárias. A gravação ocorre após o retorno da resposta ao usuário, via chamada HTTP assíncrona a um workflow secundário dedicado.

Camada de ETL Serverless: Responsável por transformar os dados brutos do Data Lake em registros analíticos estruturados. Implementada como função AWS Lambda em Python, é ativada automaticamente pelo evento de upload no S3 e produz três tabelas: `messages`, `agents\_plan` e `agents\_actions`, gravadas no bucket de Data Warehouse com particionamento Hive-style.

Camada Analítica: Responsável por viabilizar consultas sobre o histórico completo de interações. Utiliza o Amazon Athena, que executa SQL diretamente sobre os arquivos particionados no Data Warehouse, sem cluster dedicado, com custo proporcional ao volume de dados escaneado por consulta.

3.3 Camada de Ingestão

A camada de ingestão constitui o ponto único de entrada do sistema e tem como função receber requisições provenientes de qualquer canal — WhatsApp, Telegram, widget Web ou chamadas diretas via API — e normalizá-las em um objeto de dados homogêneo antes de encaminhá-las à camada de inteligência. Sua responsabilidade é estritamente delimitada: validar a autenticidade da requisição, interpretar o conteúdo multimodal recebido e garantir que o sistema multi-agentes receba sempre um contrato de dados uniforme, independentemente da origem. Nenhuma lógica de negócio ou persistência é executada nesta camada, o que a qualifica como instância direta do princípio de separação de responsabilidades enunciado por Dijkstra (1974).

A tecnologia central é o n8n, plataforma de orquestração de fluxos de trabalho executada em servidor privado virtual (VPS), que expõe um endpoint HTTP via webhook. Esse endpoint constitui a fronteira entre o mundo externo e o ecossistema de agentes, operando como um contrato de interface: qualquer canal que respeite o schema de entrada documentado na Seção 3.3.1 e apresente o token de autenticação válido descrito na Seção 3.3.2 terá sua requisição processada de forma idêntica, sem que o sistema precise conhecer ou adaptar-se ao protocolo específico do canal de origem.

3.3.1 Contrato de Dados de Entrada (Schema JSON)

A agnósticidade de canal — isto é, a capacidade de o sistema processar mensagens de WhatsApp, Telegram, widget Web ou API direta de forma indistinta — é obtida por meio de um contrato de dados explícito: um schema JSON padronizado que todo cliente deve respeitar ao submeter uma requisição ao webhook do n8n. A adoção de um contrato único em vez de adaptar a camada de inteligência a cada protocolo de origem é uma materialização direta do princípio de separação de responsabilidades (DIJKSTRA, 1974): o Orquestrador não precisa conhecer a proveniência da mensagem, apenas consumir o objeto JSON que lhe é entregue com estrutura garantida.

A requisição enviada ao sistema é composta por quatro campos de nível raiz, descritos no Quadro 3.1.

Quadro 3.1 — Schema JSON dos campos enviados na requisição de entrada

Campo	Tipo	Obrigatório	Descrição
user	string	Sim	Nome ou identificador do remetente. Utilizado para personalização das respostas e para identificação dos registros no Data Warehouse.
sessionId	string	Sim	Identificador único da sessão de conversa. Utilizado pelo Orquestrador para recuperar o histórico de contexto armazenado em Redis, garantindo continuidade entre turnos de uma mesma interação.
source	string	Sim	Canal de origem da mensagem (ex.: "wpp" para WhatsApp). Não padronizado de forma rígida pelo sistema; serve como metadado livre de identificação nas consultas analíticas do Data Warehouse.
input	array	Sim	Lista de itens de conteúdo que compõem a mensagem. Cada elemento é um objeto JSON com estrutura dependente da modalidade do conteúdo (texto ou imagem), detalhada nos Quadros 3.2 e 3.3.

Fonte: elaborado pelo autor.

Os campos `time_msg_start` e `time_processing_finished`, presentes nos registros do Data Warehouse, não fazem parte da requisição de entrada: eles são coletados internamente pelo próprio sistema — o primeiro no momento em que o webhook do n8n recebe a mensagem, e o segundo ao término do processamento pelos agentes. Essa coleta interna garante que os timestamps sejam homogêneos e independentes do relógio do cliente.

O campo `input` é o núcleo do contrato de dados: um array de objetos JSON que representa o conteúdo da mensagem, podendo combinar itens textuais e visuais em uma única

requisição. A estrutura de cada objeto varia conforme a modalidade do conteúdo. Para itens textuais, o schema é descrito no Quadro 3.2.

Quadro 3.2 — Schema do objeto textual no array input

Campo	Tipo	Valor	Descrição
type	string	"txt"	Identifica o item como conteúdo textual.
mimeType	string	"plainText"	Tipo MIME do conteúdo. Para itens textuais, sempre igual a "plainText".
msg	string	texto livre	Conteúdo textual da mensagem enviada pelo usuário.

Fonte: elaborado pelo autor.

Para itens de imagem, o objeto apresenta quatro campos, incluindo uma URL da mídia e uma mensagem textual de contexto associada, conforme descrito no Quadro 3.3.

Quadro 3.3 — Schema do objeto de imagem no array input

Campo	Tipo	Valor	Descrição
type	string	"image"	Identifica o item como conteúdo visual.
mimeType	string	"image/jpeg"	Tipo MIME da imagem. Para a implementação atual, sempre igual a "image/jpeg".
img_url	string	URL	Endereço HTTP da imagem a ser processada. Na etapa de pré-processamento, o sistema recupera a mídia a partir dessa URL e a converte para o formato Base64 antes de encaminhá-la aos agentes de visão.
msg	string	texto livre	Mensagem textual de contexto associada à imagem. Permite que o remetente forneça, simultaneamente, uma pergunta ou instrução sobre o conteúdo visual enviado.

Fonte: elaborado pelo autor.

A coexistência de `img_url` e `msg` no mesmo objeto de imagem é uma decisão de design que permite ao remetente enviar, em uma única requisição, tanto o conteúdo visual quanto a pergunta ou instrução associada a ele — sem que o sistema precise inferir o contexto da imagem a partir de mensagens anteriores. A Figura 3.3 apresenta um exemplo real de payload de entrada capturado durante a fase de implantação do sistema.

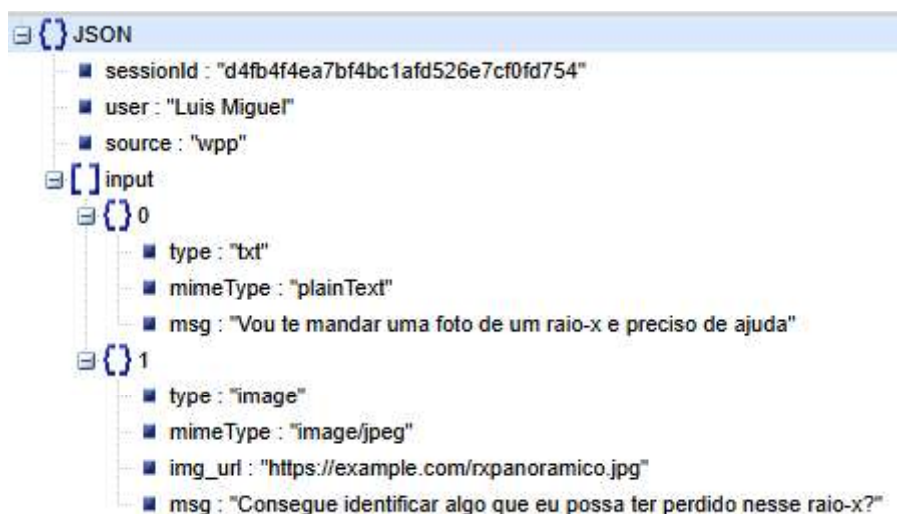


Figura 3.3 — Exemplo real de payload de entrada capturado na fase de implantação. Fonte: log de interação real coletado pelo sistema (2026).

3.3.2 Autenticação via Bearer Token

Antes de qualquer processamento de conteúdo, o webhook do n8n executa uma etapa de validação de identidade: toda requisição recebida deve apresentar, no cabeçalho HTTP Authorization, uma chave estática previamente configurada. Requisições que não apresentem o cabeçalho ou que apresentem valor divergente são rejeitadas imediatamente com código de status 401 Unauthorized, sem que o payload seja inspecionado ou encaminhado à camada de inteligência. Esse mecanismo é classificado como autenticação por token de portador (*bearer token*), modelo no qual a posse do token é suficiente para autenticar a requisição, conforme especificado pelo Internet Engineering Task Force (IETF, 2012).

A relevância dessa camada de segurança é amplificada pelo contexto de operação do sistema. O ecossistema de agentes descrito neste trabalho trata informações de natureza clínica — descrições de casos odontológicos, orientações sobre medicamentos, dados de pacientes e de profissionais de saúde. O controle de acesso no ponto de entrada é, portanto, a primeira linha de defesa contra o uso indevido do sistema e contra a exposição não autorizada dessas informações.

Do ponto de vista técnico, a escolha pela transmissão do token via cabeçalho HTTP — em detrimento de alternativas como parâmetros de URL ou campos no corpo da requisição — impede que a credencial seja exposta em logs de servidor, históricos de navegador ou no campo

Referer de requisições subsequentes, todos vetores clássicos de vazamento de credenciais em APIs HTTP (OWASP, 2023).

3.3.3 Pré-processamento Multimodal — Separação de Texto e Imagem

Após a validação do token, o n8n executa uma etapa de pré-processamento que percorre o array input e classifica cada item por modalidade antes de qualquer agente ser ativado. A lógica de classificação inspeciona os campos type e mimeType de cada objeto: itens com mimeType igual a "plainText" são identificados como conteúdo textual e seu campo msg é extraído diretamente; itens com mimeType igual a "image/jpeg" são identificados como conteúdo visual — a URL contida em img_url é utilizada para recuperar a mídia, que é convertida para o formato Base64 e combinada com o texto de contexto presente em msg antes de ser encaminhada ao agente pertinente.

A conversão antecipada de URL para Base64 é necessária porque os modelos de linguagem com capacidade de visão, utilizados pelos Agentes A a F, consomem imagens como dados incorporados na requisição, não como referências externas. Realizar essa conversão na camada de ingestão, em vez de dentro de cada agente que possa precisar de conteúdo visual, elimina a duplicação de lógica e garante que os agentes recebam os dados já no formato que consomem.

O resultado do pré-processamento é a divisão do conteúdo original em dois fluxos independentes: um contendo a concatenação de todos os itens textuais da mensagem, e outro contendo as imagens em Base64 acompanhadas de seus respectivos contextos textuais. Essa separação produz efeitos em dois momentos distintos do pipeline. Na camada de inteligência, ela permite que o Orquestrador tome decisões de roteamento com base na modalidade do conteúdo presente: uma mensagem que contenha ao menos um item de imagem ativa o Agente A como etapa anterior à classificação de intenção — cabendo a ele determinar qual dos Agentes B a F é o mais adequado para tratar aquele conteúdo visual —, enquanto mensagens exclusivamente textuais são roteadas diretamente ao agente especialista pertinente.

Na camada analítica, a separação é preservada no schema do Data Warehouse, que registra individualmente os campos is_text_question, question_text, is_image_question e question_image para cada interação — estrutura que viabiliza a análise comparativa de latência entre entradas textuais e multimodais, detalhada na Seção 3.7.

3.4 Camada de Inteligência — Sistema Multi-Agentes

A camada de inteligência constitui o núcleo funcional do sistema proposto. É nela que a mensagem recebida pela camada de ingestão é processada, interpretada e respondida. Sua implementação materializa o conceito de Sistema Multi-Agentes (MAS), paradigma no qual a inteligência emerge não de um único módulo monolítico, mas da coordenação estruturada entre múltiplos agentes especializados, cada um operando sobre um domínio restrito de conhecimento (WOOLDRIDGE; JENNINGS, 1995). As subseções a seguir descrevem o padrão arquitetural adotado, o papel de cada componente no ecossistema e os mecanismos de roteamento, memória e avaliação de qualidade que garantem o funcionamento coerente do conjunto.

3.4.1 Padrão Orchestrator-Workers aplicado

A organização interna da camada de inteligência segue o padrão arquitetural denominado Orchestrator-Workers, no qual um agente central — o Orquestrador — detém a visão completa da tarefa recebida e delega subtarefas a agentes especialistas, denominados workers. Os workers não se comunicam entre si diretamente; toda troca de informação passa obrigatoriamente pelo Orquestrador, que é o único componente com acesso ao estado global da interação e com responsabilidade pela consolidação da resposta final.

A adoção do modelo hierárquico é uma decisão deliberada: ela garante que toda decisão de roteamento seja centralizada no Orquestrador, tornando o fluxo decisório auditável de ponta a ponta — pré-requisito para a análise de comportamento do sistema no Data Warehouse, descrita na Seção 3.7.2.

Do ponto de vista técnico, o padrão é implementado no n8n por meio de dois tipos distintos de nó LangChain. O Orquestrador é configurado como um nó do tipo agent, que expõe um loop de raciocínio completo: recebe o input, delibera sobre qual ação tomar e executa chamadas de ferramenta até produzir uma resposta final. Cada agente especialista é configurado como um nó do tipo agentTool — registrado no Orquestrador como uma tool disponível para invocação, com uma descrição textual que define quando deve ser utilizada (KUMAR, 2024). Essa arquitetura é consistente com o padrão ReAct (Reasoning and Acting), descrito por Yao et al. (2023), no qual o agente alterna entre passos de raciocínio explícito e passos de ação — no caso, a invocação de um agente especialista — até convergir para uma resposta satisfatória.

3.4.2 O Orquestrador — classificação de intenção e memória de sessão

O Orquestrador é o primeiro componente a receber a mensagem após o pré-processamento realizado pela camada de ingestão. Sua função não é responder ao usuário diretamente, mas classificar o contexto da interação e decidir qual agente especialista deve receber o caso. Para isso, reconhece quatro perfis de remetente — dentista, secretaria, estagiario e tecnico_lab —, identifica a categoria da demanda e determina qual especialista deve ser acionado. O output final do Orquestrador é um objeto JSON com campos fixos: resposta ao usuário, perfil do remetente, intenção classificada, categoria, complexidade e plano de agentes acionados — estrutura que alimenta diretamente as tabelas analíticas descritas na Seção 3.6.

O Orquestrador mantém memória de sessão por meio de um banco de dados Redis, com janela de contexto de 15 turnos e TTL de 3.600 segundos. A chave de sessão é formada pela concatenação do canal de origem e do identificador de sessão, garantindo isolamento entre canais distintos.

3.4.3 Roteamento por modalidade — Agente A

Toda interação que contenha ao menos um item de imagem passa obrigatoriamente pelo Agente A antes que qualquer especialista seja acionado. Sua responsabilidade é única: receber a imagem — já convertida para Base64 pela camada de ingestão — junto ao texto de contexto, interpretar o conteúdo visual e devolver ao Orquestrador um objeto JSON estruturado com o tipo da imagem, um resumo do conteúdo visual e o contexto inferido. Esse output orienta o Orquestrador a tomar a decisão de roteamento secundária com precisão — direcionando ao Agente B para imagens clínicas, ao Agente C para rótulos de insumos, ao Agente F para equipamentos de laboratório. O Agente A não produz nenhuma resposta ao usuário final.

3.4.4 Os agentes especialistas (B a G) — domínios e responsabilidades

Após o roteamento, a mensagem é encaminhada ao agente especialista adequado. São seis agentes especializados no sistema, cada um com domínio de atuação bem delimitado, conforme apresentado no Quadro 3.4.

Quadro 3.4 — Agentes especialistas: domínio, usuário primário e tipo de tarefa

Agente	Denominação	Usuário primário	Domínio de atuação
B	Especialista Clínico	Dentista, Estagiário	Segunda opinião clínica, dosagem de medicamentos, interpretação de exames de imagem
C	Visão Técnica	Estagiário, Técnico de laboratório	Leitura de rótulos de insumos, prints de erro de equipamentos, identificação de materiais
D	Comercial	Dentista, Secretária	Estratégias de follow-up, mensagens de conversão, dúvidas comerciais
E	Atendimento	Secretária	Suporte a situações difíceis no atendimento ao paciente via WhatsApp
F	Técnico de Laboratório	Técnico de laboratório	Parâmetros de impressão 3D, fluxo de trabalho no Exocad, armazenamento de materiais
G	Documentalista	Dentista, Secretária, Estagiário	Geração de termos de consentimento odontológicos e documentos com linguagem jurídica

Fonte: elaborado pelo autor.

O Agente B atende dentistas e estagiários com dúvidas clínicas — indicação de procedimentos, conduta em casos complexos, dosagem de medicamentos. Todas as respostas reforçam que a análise é de suporte e a decisão final cabe ao profissional.

O Agente C é responsável por imagens não-clínicas: rótulos de insumos, embalagens e prints de equipamentos. Extrai dados estruturados visíveis na imagem e instrui o agente a não realizar suposições além do que é legível.

O Agente D apoia dentistas e secretárias no follow-up comercial de pacientes com orçamento não fechado, elaborando mensagens de acompanhamento personalizadas com base no contexto da consulta.

O Agente E auxilia secretárias a responder mensagens difíceis no WhatsApp — reclamações, insatisfações, situações emocionalmente delicadas —, equilibrando acolhimento ao paciente e preservação institucional da clínica.

O Agente F atende técnicos de prótese com dúvidas operacionais sobre impressão 3D odontológica, configurações no Exocad e armazenamento de materiais fotossensíveis.

O Agente G gera termos de consentimento odontológicos personalizados a partir da descrição da situação, em linguagem jurídica adequada e formato pronto para impressão e assinatura.

A granularidade de especialização entre esses seis agentes viabiliza a atribuição de modelos distintos conforme a complexidade da tarefa — estratégia detalhada na Seção 3.4.6 —, permitindo equilibrar capacidade de raciocínio e custo de inferência ao longo do ecossistema.

3.4.5 O loop de self-reflection — Agente H e o schema de avaliação

O Agente H ocupa uma posição estruturalmente distinta dos demais: ele não atende demandas do usuário, mas avalia a qualidade de toda resposta produzida pelo sistema. Seu acionamento é obrigatório e ocorre uma única vez ao final de cada interação — após todos os especialistas necessários terem concluído sua execução —, independentemente de quais agentes tenham sido utilizados no percurso.

Esse mecanismo corresponde ao padrão de self-reflection formalizado por Shinn et al. (2023) no framework Reflexion. O Agente H recebe a mensagem original do remetente e a resposta gerada pelo especialista, e produz uma avaliação estruturada com dois componentes principais.

O primeiro é uma pontuação em cinco dimensões, cada uma avaliada em escala de 0 a 10, conforme o Quadro 3.5.

Quadro 3.5 — Dimensões de avaliação do Agente H e seus critérios

Dimensão	Critério de avaliação
clareza	Qualquer usuário entenderia sem precisar perguntar novamente?
limites_respeitados	A resposta evitou diagnóstico definitivo, promessa de resultado ou compromisso jurídico indevido?
suporte	A resposta faz sentido considerando o contexto da conversa e a pergunta feita?
precisao_tecnica	A informação é factualmente correta no contexto odontológico?
fonte_identificada	A resposta distingue claramente quando a informação é uma estimativa técnica versus um dado verificado?

Fonte: elaborado pelo autor.

O segundo componente é um checklist de alucinação com três flags binários: `afirmacao_sem_base`, `contradicao` e `promessa_resultado`. A natureza binária é deliberada: qualquer ocorrência representa um risco concreto no contexto odontológico, onde afirmações incorretas sobre dosagem ou indicação têm consequência direta na segurança do paciente.

Com base nessa avaliação, o Agente H preenche o campo `requer_revisao_humana` no output. O mecanismo de acionamento de revisão humana com base nesse campo está previsto na arquitetura como evolução futura, mas não integra o escopo de implementação do presente trabalho. O campo é, no entanto, registrado e consultável via SQL no Amazon Athena, tornando-se base para análise posterior. Como o output completo do Agente H é persistido no Data Lake e normalizado pela Lambda de ETL para a tabela `agents_actions`, todas as

pontuações e flags de cada interação tornam-se evidência quantitativa auditável nos resultados deste trabalho.

3.4.6 Gestão de modelos de linguagem via OpenRouter

O sistema não está acoplado a um único provedor ou modelo de linguagem. Essa agnósticidade é obtida por meio do OpenRouter, plataforma que atua como camada de abstração sobre múltiplos provedores de LLM, expondo uma interface de API unificada compatível com o padrão de chamada da OpenAI. A atribuição de modelos segue o critério de adequação entre complexidade da tarefa e capacidade necessária do modelo, conforme o Quadro 3.6.

Quadro 3.6 — Atribuição de modelos de linguagem por agente

Agente	Modelo	Identificador OpenRouter	Justificativa de atribuição
Orquestrador	Gemini 3 Flash Preview	google/gemini-3- flash-preview	Roteamento, classificação de intenção e síntese de resposta final
Agente A	Gemma 4 26B-A4B-IT	google/gemma-4- 26b-a4b-it	Capacidade multimodal nativa para leitura e classificação de imagens
Agente B	Gemini 3 Flash Preview	google/gemini-3- flash-preview	Raciocínio clínico especializado e análise de exames de imagem
Agente C	Gemma 4 26B-A4B-IT	google/gemma-4- 26b-a4b-it	Capacidade multimodal para extração de dados de rótulos e prints de tela
Agente D	GPT 5.4 Nano	openai/gpt-5.4-nano	Geração de mensagens comerciais contextualizadas
Agente E	GPT 5.4 Nano	openai/gpt-5.4-nano	Redação de respostas de atendimento ao paciente
Agente F	Qwen 3.6 Plus	qwen/qwen3.6-plus	Raciocínio técnico-especializado para parâmetros de laboratório
Agente G	Qwen 3.6 Plus	qwen/qwen3.6-plus	Geração de documentos com precisão jurídica e terminologia formal

Agente H	GPT Nano	5.4	openai/gpt-5.4-nano	Avaliação crítica estruturada e detecção de alucinação
----------	-------------	-----	---------------------	---

Fonte: elaborado pelo autor.

Como o OpenRouter registra o consumo de tokens e o custo de inferência por requisição, é possível extrair da própria plataforma o custo real atribuído a cada agente individualmente ao final do período de implantação — sem instrumentação adicional no código ou na infraestrutura. Essa análise sobre dados reais de produção responde à pergunta de pesquisa relacionada à viabilidade econômica enunciada na tese.

3.5 Camada de Persistência — Data Lake

A camada de persistência constitui o repositório primário de dados brutos do sistema. É nela que cada interação processada pela camada de inteligência é serializada em formato JSON e armazenada de forma integral e imutável no Amazon S3 — sem transformações, filtrações ou perdas de informação. Essa abordagem segue o princípio fundamental de data lakes, conforme definido por Nargesian et al. (2019): armazenar dados em sua forma original, preservando o máximo de contexto possível para que transformações analíticas possam ser aplicadas a posteriori, sem que a ausência de um esquema rígido na ingestão comprometa a completude do dado de origem. No sistema proposto, esse princípio é operacionalmente relevante porque o JSON bruto inclui não apenas a resposta final ao usuário, mas também o raciocínio intermediário do Orquestrador, as chamadas individuais a cada agente especialista e os timestamps de início e término da interação — informações que seriam descartadas por um sistema de persistência orientado apenas ao resultado final. Todo o pipeline analítico descrito na Seção 3.7 tem como único insumo os arquivos produzidos por esta camada.

3.5.1 Desacoplamento entre inferência e persistência

Uma das decisões arquiteturais mais relevantes do sistema é a separação temporal e estrutural entre a entrega da resposta ao usuário e a gravação dos dados da interação. No fluxo implementado, ao final da execução do pipeline de agentes, o n8n realiza duas operações independentes: retorna a resposta ao canal de origem — WhatsApp, Telegram ou widget web — e, em seguida, dispara uma chamada HTTP POST de forma assíncrona para um segundo workflow dedicado exclusivamente à persistência no S3, exposto no endpoint `api/v1/save-data`.

Do ponto de vista do usuário, a interação está encerrada antes que a gravação sequer seja iniciada — ambas as operações correm em paralelo, mas o retorno ao canal não aguarda a confirmação do upload.

Esse modelo de desacoplamento manifesta diretamente os dois princípios arquiteturais estabelecidos na Seção 3.1. Do ponto de vista da Separação de Responsabilidades (DIJKSTRA, 1974), a camada de inteligência não contém nenhuma lógica de persistência; ela delega essa responsabilidade integralmente ao workflow de salvamento, comunicando-se com ele exclusivamente por meio de um contrato explícito — o payload JSON enviado via HTTP. Do ponto de vista da Arquitetura Orientada a Eventos (MICHELSON, 2006), a conclusão do pipeline de agentes constitui o evento produtor: o workflow de inteligência conhece apenas o fato de que a interação terminou e precisa ser registrada, sem qualquer conhecimento sobre como o armazenamento é implementado.

Em termos de resiliência, o desacoplamento garante que uma eventual falha no processo de persistência — indisponibilidade temporária da API do S3, erro de validação do token de autorização ou timeout de rede — não impacte a experiência do usuário, que já recebeu sua resposta. O dado que eventualmente não foi gravado pode ser reprocessado a partir do log interno do n8n sem necessidade de reexecutar o pipeline de agentes. Comparativamente, em arquiteturas síncronas — nas quais inferência e gravação ocorrem na mesma cadeia de execução —, uma latência adicional na operação de escrita é diretamente somada ao tempo de resposta percebido pelo usuário a cada interação.

O workflow de persistência inclui uma etapa de autenticação por token estático, verificado tanto no cabeçalho HTTP Authorization quanto como parâmetro de query string, garantindo que apenas chamadas originárias do workflow principal sejam aceitas. Apenas após a validação bem-sucedida do token o fluxo avança para as etapas de construção do caminho de armazenamento e upload do arquivo.

3.5.2 Organização do S3

Os arquivos são armazenados no bucket `blanvur-data-lake`, organizado em pastas nomeadas pela data de criação no formato `DD-MM-YYYY`. A cada execução do workflow de persistência, o sistema extrai os componentes da data corrente — dia, mês e ano —, constrói o nome da pasta no formato especificado e verifica, por meio de uma consulta ao S3, se essa pasta

já existe. Caso não exista, ela é criada automaticamente antes do upload. Esse comportamento garante que a estrutura de diretórios do bucket seja autossustentável e não requeira provisionamento manual — propriedade alinhada ao modelo serverless descrito na Seção 3.1.3.

O nome de cada arquivo é composto pela concatenação de quatro campos separados por hífen: `{timestamp}-{source}-{sessionId}-{user}.json`. O campo `timestamp` corresponde ao instante exato de execução do workflow de persistência, expresso em milissegundos, garantindo unicidade mesmo em sessões simultâneas iniciadas no mesmo segundo. O campo `source` identifica o canal de origem da mensagem — por exemplo, `chatbutton` para o widget web —, permitindo segmentar o volume de interações por canal diretamente a partir do nome do arquivo. O campo `sessionId` é o identificador único de sessão gerado pelo n8n, que associa todos os arquivos de uma mesma conversa contínua. O campo `user` corresponde ao nome do remetente com espaços removidos, viabilizando a identificação do operador sem necessidade de abrir o arquivo.

Essa convenção de nomenclatura produz arquivos autoexplicativos: canal, sessão, usuário e momento da gravação são legíveis diretamente na listagem do bucket, sem necessidade de desserialização do conteúdo. Isso é especialmente relevante em cenários de auditoria manual ou reprocessamento seletivo — situações nas quais um analista precisa identificar rapidamente um subconjunto de arquivos sem recorrer a consultas SQL.

3.5.3 Schema do JSON bruto persistido

Cada arquivo JSON gravado no S3 representa uma interação completa e autocontida. O objeto raiz do arquivo é composto por três blocos estruturalmente distintos — `output`, `steps` e os metadados da sessão —, cujos campos estão documentados nos Quadros 3.7, 3.8 e 3.9, respectivamente.

O bloco `output` registra o resultado da deliberação do Orquestrador após o processamento da mensagem. Seus campos são apresentados no Quadro 3.7.

Quadro 3.7 — Campos do bloco output

Campo	Tipo	Descrição
resposta	string	Texto final consolidado a ser entregue ao usuário
perfil_remetente	string	Perfil inferido pelo Orquestrador (dentista, secretaria, estagiario, tecnico_lab)
intencao	string	Classificação textual da intenção detectada na mensagem
categoria	string	Domínio funcional da demanda (comercial, clinico, laboratorio, etc.)
complexidade da resposta	string	Estimativa qualitativa de complexidade (baixa, média, alta)
plano	array	Lista ordenada dos agentes acionados, cada um com os campos agente e instrucao
acionar_auditor	boolean	Indica se o Agente H foi incluído no plano de execução

Fonte: elaborado pelo autor.

O bloco steps é um array que registra cada chamada de ferramenta executada pelo Orquestrador durante o pipeline. Cada elemento do array corresponde a uma invocação de agente e contém os campos apresentados no Quadro 3.8.

Quadro 3.8 — Campos de cada elemento do array steps

Campo	Tipo	Descrição
action.tool	string	Nome do agente invocado (ex.: agente_d, agente_h)
action.toolInput.Prompt__User_Message	string	Instrução completa enviada pelo Orquestrador ao agente
action.log	string	Log interno de raciocínio produzido pelo modelo de linguagem antes de executar a chamada
action.messageLog	array	Histórico serializado da cadeia de mensagens do LangChain para aquela chamada
observation	string	Output estruturado retornado pelo agente após sua execução, em formato JSON serializado como string

Fonte: elaborado pelo autor.

Por fim, os metadados da sessão estão distribuídos nos campos de nível raiz do objeto JSON, conforme o Quadro 3.9.

Quadro 3.9 — Metadados de sessão

Campo	Tipo	Descrição
sessionId	string	Identificador único da sessão gerado pelo n8n
source	string	Canal de origem da mensagem (ex.: chatbutton)
user	string	Nome do remetente conforme registrado no canal
input	array	Lista de objetos representando o conteúdo original da mensagem; cada objeto contém type (modalidade), mimeType e msg (conteúdo textual ou URL de imagem)
time_msg_start	string (ISO 8601)	Timestamp de recebimento da mensagem pelo n8n
time_processing_finished	string (ISO 8601)	Timestamp de conclusão do pipeline de agentes; a diferença entre este campo e time_msg_start constitui a latência total da interação

Fonte: elaborado pelo autor.

A estrutura do array input merece atenção específica: ao preservar cada item de entrada individualmente — com sua modalidade e tipo MIME —, o schema permite à camada de ETL, descrita na Seção 3.6, distinguir com precisão quais interações continham exclusivamente texto e quais continham imagens, sem ambiguidade. Essa distinção é condição necessária para o cálculo da métrica de latência multimodal apresentada na Seção 3.7.

3.6 Camada de Processamento — ETL Serverless

A camada de processamento tem por função transformar os registros brutos depositados no Data Lake em dados analíticos normalizados, estruturados e consultáveis. Ela é implementada como uma função AWS Lambda escrita em Python, acionada automaticamente a cada novo arquivo .json criado no bucket *blanvur-data-lake*, sem qualquer chamada explícita ou agendamento externo. Cada execução produz três arquivos JSON gravados no bucket *blanvur-data-warehouse*, correspondentes às três tabelas analíticas do sistema: *messages*, *agents_plan* e *agents_actions*. A função desconhece os agentes, os modelos de linguagem e o n8n — ela opera exclusivamente sobre o contrato de dados do JSON de entrada —, o que a qualifica como instância direta do princípio de Separação de Responsabilidades (SoC) enunciado por Dijkstra (1974): cada componente do sistema concentra uma única preocupação bem definida e se comunica com os demais por meio de contratos explícitos.

3.6.1 Trigger automático via evento S3

O mecanismo de acionamento da Lambda fundamenta-se no serviço de notificações de eventos do Amazon S3. Toda vez que o workflow de persistência do n8n realiza o upload de um arquivo .json no bucket do Data Lake, o S3 emite automaticamente um evento do tipo *s3:ObjectCreated:** e o encaminha à função Lambda previamente configurada como destino daquela notificação. O evento carrega o nome do bucket e a chave do objeto recém-criado, informações suficientes para que a Lambda localize, leia e processe o arquivo. Nenhuma chamada explícita, nenhum agendamento periódico e nenhuma instrumentação adicional no n8n são necessários: o ato de depositar o arquivo no bucket é, em si, o único gatilho do pipeline de transformação.

Esse mecanismo materializa o paradigma da Arquitetura Orientada a Eventos (EDA) na camada de dados do sistema. Conforme definido por Michelson (2006), numa EDA o fluxo de execução é determinado pela ocorrência de eventos, e o produtor conhece apenas o fato de que o evento ocorreu — sem qualquer conhecimento sobre quem o consumirá ou como processará a informação. No sistema proposto, o workflow de persistência do n8n (produtor) conhece unicamente que precisa realizar o upload do JSON; a Lambda (consumidora) conhece unicamente que deve processar o arquivo cujo caminho lhe foi entregue pelo evento. Ambos

operam de forma autônoma, conectados exclusivamente pelo evento de upload e pelo contrato de dados do arquivo JSON.

Resiliência e garantias de reprocessamento. O desacoplamento entre produtor e consumidora produz garantias de resiliência que seriam inviáveis em arquiteturas síncronas. Em primeiro lugar, o dado de origem permanece intacto e imutável no bucket do Data Lake independentemente do resultado da execução da Lambda. Se um arquivo for malformatado, se a Lambda atingir o *timeout* de execução configurado ou se houver indisponibilidade temporária do bucket de destino, o arquivo original não é alterado nem removido — ele continua disponível para reprocessamento manual ou automático sem necessidade de reexecutar o pipeline de agentes que o gerou. Essa propriedade está alinhada ao princípio descrito por Kleppmann (2017) segundo o qual sistemas orientados a dados devem armazenar o evento original de forma imutável, permitindo que qualquer transformação analítica seja reaplicada sobre o dado bruto quantas vezes forem necessárias.

Em segundo lugar, a Lambda implementa tratamento de erros por registro individual: eventuais falhas no processamento de um item específico dentro de um arquivo JSON são capturadas, registradas em log e acumuladas em uma lista de erros, sem interromper o processamento dos demais registros do mesmo arquivo. Ao término da execução, a função retorna o código HTTP 207 Multi-Status quando há erros parciais, distinguindo explicitamente o resultado de uma execução parcialmente bem-sucedida de uma falha total. Esse comportamento garante que uma anomalia em um único registro não invalide o processamento de uma ingestão contendo múltiplos registros.

Em terceiro lugar, a função verifica, como primeira etapa do processamento de cada objeto, se a chave recebida possui extensão .json. Objetos de outros tipos — como arquivos de marcação .keep utilizados para sinalizar partições vazias — são ignorados com registro em log, sem que a função retorne erro. Esse filtro previne que artefatos internos do próprio pipeline de ETL acionem reprocessamentos desnecessários em loop.

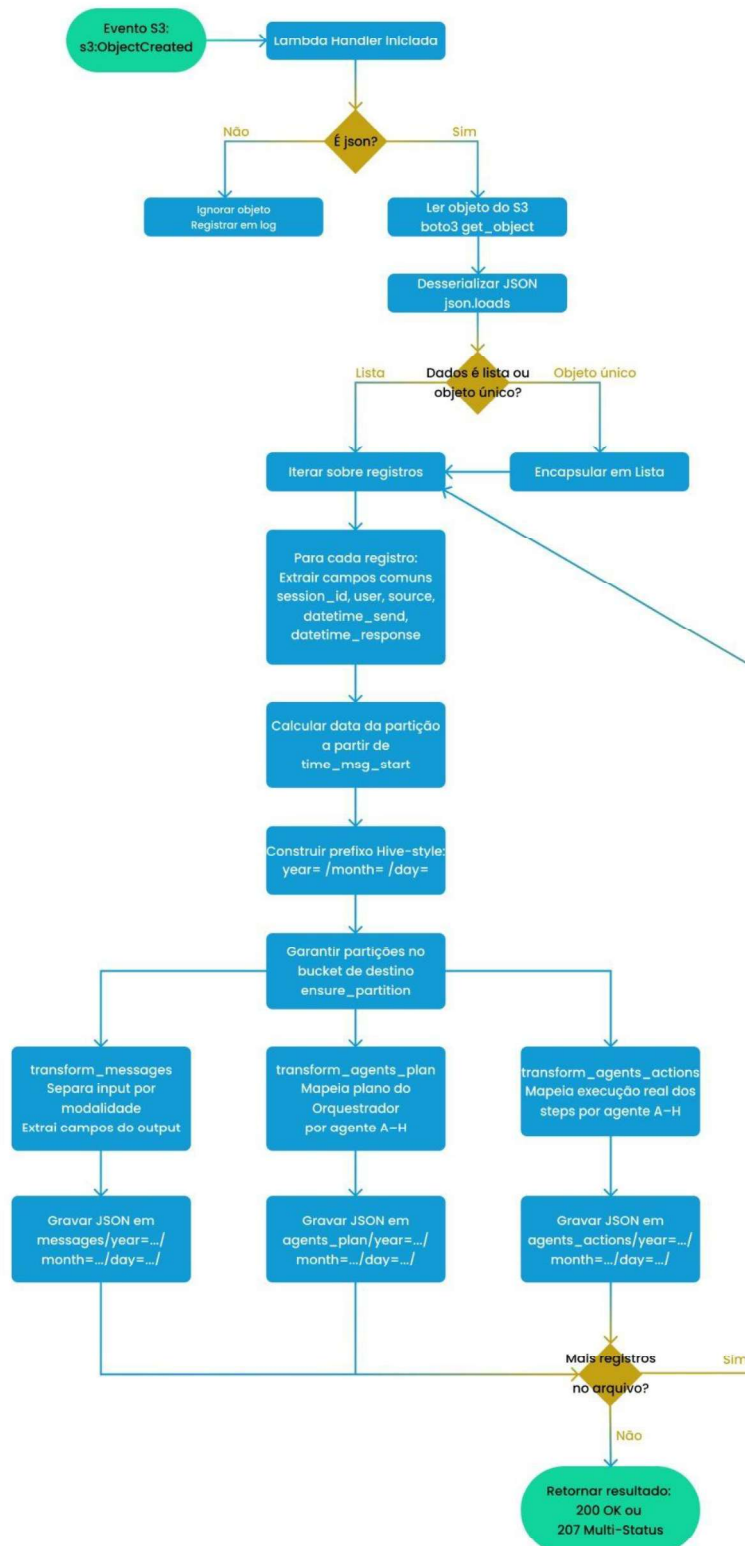
Wang et al. (2018) identificam que o modelo serverless apresenta custo zero em períodos sem atividade e escalabilidade automática sob demanda. Do ponto de vista de resiliência, essas propriedades implicam que o sistema de ETL nunca precisa ser provisionado para o pico de volume de interações: cada evento S3 gera uma invocação independente da

Lambda, e a AWS gerencia o paralelismo sem configuração adicional. Em cenários de ingestão simultânea — como o encerramento concentrado de sessões ao final do expediente da clínica —, múltiplas instâncias da Lambda são executadas concorrentemente sem contenção entre si, uma vez que cada instância opera sobre um arquivo distinto e grava em partições já garantidas ou criadas de forma isolada.

3.6.2 As três tabelas de destino e suas transformações

Para cada registro processado, a Lambda executa três funções de transformação independentes e grava os resultados em tabelas distintas no bucket do Data Warehouse. Essa separação em três tabelas não é arbitrária: ela reflete três níveis de granularidade da mesma interação, do mais agregado — a mensagem e sua resposta final — ao mais granular — os passos internos de cada agente. O fluxo completo de execução da função, desde o recebimento do evento S3 até a escrita dos três arquivos de saída, está ilustrado na Figura 3.4.

Figura 3.4 — Fluxograma de execução da função AWS Lambda de ETL



Fonte: elaborado pelo autor com base no código-fonte da função Lambda (2026).

As três tabelas compartilham um conjunto de cinco campos comuns, extraídos pela função `extract_common_fields` a partir do JSON de origem: `session_id`, `user`, `source`, `datetime_send` e `datetime_response`. Esses campos funcionam como chave de junção entre as tabelas em consultas SQL no Athena, permitindo correlacionar o resultado final de uma interação com o plano de agentes que a originou e com as ações efetivamente executadas em cada passo. A seguir, cada tabela é descrita em detalhe quanto à sua função analítica e ao processo de transformação que a produz, com os campos documentados nos Quadros 3.10, 3.11 e 3.12.

Tabela messages

A tabela `messages` registra o resultado consolidado de cada interação do ponto de vista do usuário final. A função `transform_messages` percorre o array `input` do JSON bruto e classifica cada elemento por modalidade, inspecionando os campos `type` e `mimeType`: objetos com `type` igual a `txt` ou `mimeType` igual a `plainText` têm o campo `msg` extraído e concatenado como conteúdo textual; objetos com `type` igual a `image` têm o campo `msg` — o texto de contexto associado à imagem — separado como conteúdo visual. O resultado são quatro campos que descrevem a natureza da pergunta original: `is_text_question` e `question_text` para o componente textual, `is_image_question` e `question_image` para o componente visual. Adicionalmente, a função extrai os campos do bloco `output` do JSON bruto, que registram a deliberação final do Orquestrador. Essa tabela é o insumo primário para as análises de latência, volume de uso e distribuição de categorias descritas na Seção 3.7.

Quadro 3.10 — Estrutura da tabela messages

Campo	Tipo	Origem no JSON bruto	Descrição
session_id	string	\$.sessionId	Identificador único da sessão de conversa
user	string	\$.user	Nome do remetente conforme registrado no canal
source	string	\$.source	Canal de origem da mensagem (ex.: chatbutton, wpp)
datetime_send	string (ISO 8601)	\$.time_msg_start	Timestamp de recebimento da mensagem pelo n8n
datetime_response	string (ISO 8601)	\$.time_processing_finished	Timestamp de conclusão do pipeline de agentes
identified_profile	string	\$.output.perfil_remetente	Perfil do remetente inferido pelo Orquestrador

category	string	\$.output.categoria	Domínio funcional da demanda (ex.: comercial, clinico)
complexity_question	string	\$.output.complexidade da resposta	Estimativa qualitativa de complexidade
intention	string	\$.output.intencao	Classificação textual da intenção detectada
response	string	\$.output.resposta	Texto final consolidado entregue ao usuário
is_text_question	boolean	Derivado de \$.input[].type	Indica se a mensagem continha componente textual
question_text	string	Derivado de \$.input[].msg	Conteúdo textual da mensagem original
is_image_question	boolean	Derivado de \$.input[].type	Indica se a mensagem continha componente visual
question_image	string	Derivado de \$.input[].msg	Texto de contexto associado à imagem enviada

Fonte: elaborado pelo autor.

Tabela `agents_plan`

A tabela `agents_plan` registra o plano de execução deliberado pelo Orquestrador *antes* da ativação dos agentes — isto é, a intenção declarada pelo sistema sobre quais especialistas seriam acionados e com qual instrução. A função `transform_agents_plan` itera sobre o array `output.plano` do JSON bruto e, para cada um dos oito agentes possíveis (A a H), produz um par de campos: `agent_X_active` (booleano que indica se o agente foi incluído no plano) e `agent_X_instruction` (o texto da instrução definida pelo Orquestrador para aquele agente). O resultado é um registro de largura fixa com 16 campos de agentes, diretamente consultável por SQL sem desaninhamento de arrays.

Quadro 3.11 — Estrutura da tabela `agents_plan`

Campo	Tipo	Origem no JSON bruto	Descrição
<code>session_id</code>	string	<code>\$.sessionId</code>	Identificador único da sessão (chave de junção)
<code>user</code>	string	<code>\$.user</code>	Nome do remetente
<code>source</code>	string	<code>\$.source</code>	Canal de origem
<code>datetime_send</code>	string (ISO 8601)	<code>\$.time_msg_start</code>	Timestamp de envio
<code>datetime_response</code>	string (ISO 8601)	<code>\$.time_processing_finished</code>	Timestamp de resposta

agent_A_active	boolean	\$.output.plano[].agente	Agente A incluído no plano do Orquestrador
agent_A_instruction	string	\$.output.plano[].instrucao	Instrução definida para o Agente A
agent_B_active	boolean	\$.output.plano[].agente	Agente B incluído no plano do Orquestrador
agent_B_instruction	string	\$.output.plano[].instrucao	Instrução definida para o Agente B
(padrão repetido para C, D, E, F, G)	—	—	—
agent_H_active	boolean	\$.output.plano[].agente	Agente H (Auditor) incluído no plano
agent_H_instruction	string	\$.output.plano[].instrucao	Instrução definida para o Agente H

Nota: a tabela contém 16 campos de agentes no total — dois por agente (A a H) —, além dos cinco campos comuns de sessão. Fonte: elaborado pelo autor.

Tabela agents_actions

A tabela agents_actions registra a execução *real* de cada agente durante o pipeline — os *intermediate steps* produzidos pelo loop de raciocínio do framework LangChain. A função transform_agents_actions itera sobre o array steps do JSON bruto e agrupa os passos por agente (action.tool). Para cada agente que executou ao menos um passo, são extraídos e concatenados:

o prompt completo enviado pelo Orquestrador (toolInput.Prompt_User_Message), o log interno de raciocínio do modelo antes da chamada (action.log) e a observação estruturada retornada pelo agente após sua execução (observation). Quando um mesmo agente é invocado mais de uma vez na mesma interação, os valores são concatenados com o separador \n---\n, preservando a sequência de execução. O resultado segue estrutura análoga à tabela anterior: quatro campos por agente (active, toolInput_Prompt_User_Message, log, observation), totalizando 32 campos de agentes.

Quadro 3.12 — Estrutura da tabela agents_actions

Campo	Tipo	Origem no JSON bruto	Descrição
session_id	string	\$.sessionId	Identificador único da sessão (chave de junção)
user	string	\$.user	Nome do remetente
source	string	\$.source	Canal de origem
datetime_send	string (ISO 8601)	\$.time_msg_start	Timestamp de envio
datetime_response	string (ISO 8601)	\$.time_processing_finished	Timestamp de resposta

agent_A_actions_active	boolean	\$.steps[].action.tool	Agente A efetivamente executado durante o pipeline
agent_A_actions_toolInput_Prompt_User_Message	string	\$.steps[].action.toolInput	Prompt completo enviado ao Agente A pelo Orquestrador
agent_A_actions_log	string	\$.steps[].action.log	Log interno de raciocínio do modelo antes da chamada
agent_A_actions_observation	string	\$.steps[].observation	Output estruturado retornado pelo Agente A
<i>(padrão repetido para B, C, D, E, F, G)</i>	—	—	—
agent_H_actions_active	boolean	\$.steps[].action.tool	Agente H (Auditor) efetivamente executado
agent_H_actions_toolInput_Prompt_User_Message	string	\$.steps[].action.toolInput	Prompt enviado ao Agente H

agent_H_actions_log	string	\$.steps[].action.log	Log de raciocínio do Agente H
agent_H_actions_observation	string	\$.steps[].observation	Output do Agente H, contendo pontuações e flags de alucinação

Nota: a tabela contém 32 campos de agentes no total — quatro por agente (A a H) —, além dos cinco campos comuns de sessão. Fonte: elaborado pelo autor.

A separação entre plano e execução como instrumento analítico

A existência de duas tabelas distintas para o plano (`agents_plan`) e para a execução (`agents_actions`) é a decisão de design mais relevante desta camada para fins analíticos. O plano representa a intenção declarada do Orquestrador — os agentes que ele afirmou que acionaria e as instruções que os direcionariam. As ações representam o comportamento real do sistema — os agentes efetivamente invocados, com os prompts reais transmitidos e as respostas que produziram.

A comparação entre essas duas tabelas por meio de consultas SQL no Athena permite identificar divergências entre comportamento pretendido e comportamento observado: casos em que um agente planejado não foi executado, ou em que um agente não previsto no plano foi invocado durante o loop de raciocínio do Orquestrador, constituem indicadores de inconsistência interna do sistema agêntico diretamente mensuráveis. Essa capacidade analítica viabiliza, no âmbito deste trabalho, a verificação empírica de que a IA se comporta conforme planejado — contribuição que distingue o sistema proposto das implementações convencionais de IA conversacional, que, conforme apontado por Wooldridge e Jennings (1995), carecem de mecanismos explícitos para observação e auditoria do raciocínio interno dos agentes.

3.6.3 Particionamento Hive-style para o Athena

Os arquivos gravados no bucket do Data Warehouse seguem o padrão de particionamento denominado *Hive-style*, no qual os objetos são organizados em prefixos de caminho que codificam as dimensões da partição no formato chave=valor. Esse padrão foi originalmente introduzido pelo Apache Hive — sistema de *data warehouse* construído sobre o Hadoop — como convenção para organizar dados em diretórios que refletem os predicados de filtragem mais frequentes nas consultas analíticas (THUSOO et al., 2009). Sistemas de consulta serverless modernos, como o Amazon Athena, adotaram essa mesma convenção para inferir automaticamente as partições disponíveis sem necessidade de catálogo externo.

No sistema proposto, o prefixo adotado é `year=AAAA/month=MM/day=DD/`, calculado a partir do campo `time_msg_start` do JSON bruto — o timestamp de recebimento da mensagem pelo n8n, que define a data à qual a interação pertence. A função `build_partition_prefix` da Lambda monta esse prefixo a partir de um objeto `datetime` UTC, conforme o trecho abaixo:

```
def build_partition_prefix(dt: datetime) -> str:

    return f"year={dt.year}/month={dt.month:02d}/day={dt.day:02d}/"
```

A adoção do tempo universal coordenado (UTC) como referência de particionamento é uma decisão deliberada que preserva a consistência das partições independentemente do fuso horário do canal de origem ou do servidor de operação do n8n. Para cada tabela de destino e para cada partição ainda inexistente, a Lambda verifica a existência do prefixo via `list_objects_v2` e, quando necessário, cria um objeto marcador `.keep` no prefixo correspondente. Como o Amazon S3 não possui o conceito de diretório real — todos os objetos são armazenados em um espaço de nomes plano, e os prefixos são apenas convenções de nomenclatura —, o marcador `.keep` é o mecanismo que torna a partição visível ao Athena antes que qualquer registro de dados seja gravado nela.

Por que o particionamento importa: custo e desempenho no Athena. O Amazon Athena cobra por volume de dados escaneado em cada consulta, a uma taxa de 5 dólares por terabyte (AMAZON WEB SERVICES, 2024). Em um Data Warehouse sem particionamento, uma consulta SQL que filtre interações por uma data específica — como `WHERE datetime_send`

`>= '2026-04-18'` — precisaria varrer a totalidade dos arquivos de todas as interações históricas para localizar os registros relevantes, independentemente de quantos dados estejam fora do período de interesse. Com o particionamento Hive-style, o Athena identifica que os dados de uma data específica estão integralmente contidos no prefixo `messages/year=2026/month=04/day=18/` e limita o scan exclusivamente àquela partição, sem tocar nos demais prefixos. O custo de uma consulta filtrada por dia único torna-se, portanto, proporcional ao volume de interações daquele dia — e não ao volume histórico total.

Essa propriedade tem implicação direta na viabilidade econômica do sistema para organizações de pequeno e médio porte. Wang et al. (2018) identificam que o modelo serverless converte custos fixos de infraestrutura em custos estritamente variáveis; o particionamento é o mecanismo que garante que essa variabilidade seja proporcional ao volume de dados efetivamente necessário para responder cada consulta analítica, e não ao volume acumulado no Data Warehouse ao longo de toda a operação do sistema. Em termos práticos, para a clínica odontológica descrita neste trabalho — cujo volume de interações está concentrado em horários comerciais de dias úteis —, o particionamento diário permite que qualquer análise operacional de curto prazo (última semana, último mês) seja executada a um custo marginal muito inferior ao que seria necessário em um banco de dados relacional convencional com cobrança por capacidade provisionada.

3.7 Camada Analítica — Data Warehouse

3.7.1 Amazon Athena como motor de consulta serverless

A camada analítica do sistema não utiliza um banco de dados relacional convencional. Em vez disso, adota o Amazon Athena, serviço de consulta interativa da AWS que executa SQL diretamente sobre arquivos armazenados no Amazon S3, sem necessidade de provisionar ou gerenciar qualquer infraestrutura de banco de dados dedicada. O Athena opera sobre o princípio do schema-on-read: o esquema das tabelas é definido externamente, por meio de um catálogo de dados (AWS Glue Data Catalog), e aplicado no momento da leitura — o que permite que os arquivos JSON gravados pela função Lambda de ETL sejam imediatamente consultáveis como tabelas SQL sem qualquer etapa de importação ou carga prévia (AMAZON WEB SERVICES, 2024).

Do ponto de vista operacional, essa característica tem uma consequência direta para a observabilidade do sistema: toda interação processada pelo ecossistema de agentes torna-se consultável via SQL poucos segundos após sua conclusão, sem que qualquer componente adicional precise ser operado. O pipeline de observabilidade é, portanto, integralmente serverless: o upload no S3 dispara a Lambda, que grava os registros normalizados no bucket de Data Warehouse, que o Athena lê sob demanda.

O modelo de cobrança do Athena é igualmente relevante para os objetivos deste trabalho. O serviço cobra por volume de dados efetivamente escaneado por consulta, a uma taxa de cinco dólares por terabyte (AMAZON WEB SERVICES, 2024), sem cobrança por capacidade ociosa, por número de usuários ou por tempo de cluster ativo. Conforme demonstrado na Seção 3.6, o particionamento Hive-style adotado nas tabelas do Data Warehouse — com prefixos no formato `year=/month=/day=` — faz com que o Athena restrinja o escaneamento às partições necessárias para responder cada consulta. O custo de uma query filtrada por data específica torna-se, assim, proporcional ao volume de interações daquele dia, e não ao histórico acumulado total. Para uma clínica odontológica de pequeno ou médio porte, cujo volume de interações diárias é tipicamente inferior a algumas centenas de registros, o custo operacional da camada analítica tende a ser marginal — e, crucialmente, mensurável, o que permite incluir essa parcela no cálculo de viabilidade econômica do sistema apresentado no Capítulo 4.

3.7.2 Métricas analisáveis e justificativa de cada uma

As métricas apresentadas a seguir são derivadas das três tabelas do Data Warehouse — `messages`, `agents_plan` e `agents_actions` — e respondem diretamente às três proposições da tese enunciadas na Seção 1: que o sistema funciona, que é observável e que é economicamente viável. O Quadro 3.13 sintetiza cada métrica, a tabela que a alimenta, as colunas envolvidas e a proposição da tese à qual ela se vincula.

Quadro 3.13 — Métricas analíticas: definição, origem e proposição correspondente

Métrica	Tabela	Colunas envolvidas	Proposição
Latência por canal	messages	datetime_response datetime_send, source	– Funciona
Latência por tipo de input	messages	datetime_response datetime_send, is_image_question	– Funciona
Volume de uso diário	messages	COUNT(*), partição day=	Funciona
Taxa de acionamento por agente	agents_plan	agent_X_active (X = A a H)	Observável
Custo de inferência total	OpenRouter + messages	custo total do período ÷ COUNT(*)	Viável

Fonte: elaborado pelo autor.

Latência por canal e por tipo de input. A diferença entre `datetime_response` e `datetime_send`, ambas registradas na tabela `messages`, corresponde ao tempo total percebido pelo usuário — do momento em que a requisição chega ao webhook do n8n até o momento em que a resposta final é produzida. Segmentada pelo campo `source`, essa métrica permite comparar a latência entre canais de origem (WhatsApp, Telegram, widget Web). Segmentada pelo campo `is_image_question`, permite responder à pergunta de pesquisa específica sobre o custo de latência do processamento multimodal: interações que passam pelo Agente A tendem a apresentar latência superior às puramente textuais, e essa diferença é quantificável diretamente a partir dos dados persistidos.

Volume de uso diário. Uma contagem simples de registros na tabela `messages`, agrupada pela partição de data, produz a curva de uso do sistema ao longo do período de implantação. Essa métrica sustenta empiricamente a premissa de uso concentrado em horários comerciais, que fundamenta a justificativa da abordagem `serverless` na Seção 3.1.3: se o sistema apresentar baixíssimo volume fora do horário comercial, isso confirma que uma infraestrutura baseada em servidores dedicados implicaria custo fixo para capacidade ociosa a maior parte do tempo.

Taxa de acionamento por agente. A tabela `agents_plan` registra, para cada interação, quais agentes foram ativados pelo Orquestrador — campos `agent_A_active` a `agent_H_active`, de valor booleano. A divisão do número de ativações de cada agente pelo total de interações no período produz a taxa de acionamento. Essa métrica tem duas utilidades: operacionalmente, revela quais domínios de demanda são mais frequentes na clínica; analiticamente, serve como denominador para o cálculo de custo por agente, uma vez que agentes mais acionados têm peso maior no custo total do sistema.

Custo de inferência total. O OpenRouter consolida, em seu painel de uso, o custo total de inferência acumulado ao longo de um período — expresso em dólares, sem granularidade por sessão ou por agente individual. A métrica adotada neste trabalho é, portanto, o custo total do período de implantação, confrontado com o volume de interações registrado na tabela `messages` no mesmo intervalo. A divisão entre custo total e número de interações produz o custo médio por interação, indicador que permite avaliar diretamente a viabilidade econômica do sistema: se esse valor for inferior ao custo equivalente de atendimento humano para o mesmo volume de demandas, a proposição de sustentabilidade financeira enunciada na tese estará empiricamente sustentada.

3.7.3 Modelo de dados e orientação para análise futura

As três tabelas do Data Warehouse — `messages`, `agents_plan` e `agents_actions` — compartilham o campo `session_id` como identificador comum: toda linha que descreve a mesma interação, independentemente da tabela em que está registrada, carrega o mesmo valor nesse campo. Essa característica não é acidental — ela é resultado direto da decisão arquitetural de emitir um único objeto JSON por interação na camada de ETL, do qual as três tabelas são derivadas, conforme descrito na Seção 3.6.

No escopo deste trabalho, cada tabela é analisada de forma independente, produzindo as métricas descritas na Seção 3.7.2. O `session_id` compartilhado, no entanto, constitui a fundação técnica para a construção de dashboards analíticos consolidados em trabalhos futuros: ferramentas de Business Intelligence (BI) que consumam essas tabelas diretamente do Athena poderão correlacionar, em um único painel, dados de contexto da interação (tabela `messages`), plano de execução dos agentes (tabela `agents_plan`) e registros detalhados de cada ação executada (tabela `agents_actions`), sem que qualquer transformação adicional seja necessária na camada de dados.

3.8 Decisões de Design e Trade-offs

As escolhas tecnológicas que compõem a arquitetura proposta não são neutras: cada uma representa a preferência deliberada por um conjunto de propriedades em detrimento de outra. Esta seção documenta as principais decisões de design, explicita os critérios que as orientam e reconhece os trade-offs delas decorrentes.

3.8.1 Por que n8n como orquestrador de fluxo?

A escolha do n8n se justifica por quatro critérios combinados: suporte nativo à orquestração de agentes LLM via nós LangChain, sem exigir implementação do ciclo de raciocínio do zero (KUMAR, 2024); integração com APIs externas — WhatsApp, Telegram, OpenRouter e serviços AWS — por meio de nós prontos, sem código adicional; hospedagem própria em VPS, que mantém todas as mensagens sob controle exclusivo do operador, em conformidade com a LGPD; e interface visual que reduz o tempo entre hipótese e observação em um projeto de natureza iterativa, onde alterações de roteamento e configuração de modelos são frequentes.

Alternativas de código puro como LangGraph (LANGGRAPH, 2024) e AutoGen (WU et al., 2023) oferecem maior expressividade programática, mas exigem que cada modificação arquitetural seja implementada e implantada como código — custo de iteração que favorece sistemas em produção estável, não prototipagem de pesquisa. O trade-off dessa escolha é um overhead de execução em relação a implementações equivalentes em Python puro, aceitável dado que a latência dominante no sistema é a dos próprios modelos de linguagem.

3.8.2 Por que S3 + Athena e não um banco de dados relacional?

Um banco de dados relacional opera sob o padrão schema-on-write (KLEPPMANN, 2017): qualquer alteração na estrutura dos dados — como a adição de um campo no JSON gerado pelo Orquestrador — exige migração de schema. Para um sistema cujos outputs são semiestruturados e evoluem junto com os prompts, esse modelo impõe rigidez incompatível com o ciclo de desenvolvimento. O par S3 + Athena inverte essa lógica: os arquivos JSON são gravados sem validação de schema, e a estrutura analítica é aplicada apenas no momento da consulta (schema-on-read), permitindo que o histórico de interações já coletado nunca seja invalidado por mudanças no design dos agentes.

O segundo argumento é econômico. Um banco relacional gerenciado exige instância provisionada continuamente, com cobrança por hora independente do volume de uso. O Amazon Athena cobra pelo volume de dados escaneado por consulta, e o S3 cobra por volume armazenado — modelo que converte custo fixo em custo estritamente variável (WANG et al., 2018). Para clínicas de pequeno e médio porte, com uso concentrado em horários comerciais, isso representa viabilidade econômica diretamente mensurável — e, portanto, um resultado quantitativo deste trabalho. Adicionalmente, a carga analítica do sistema é OLAP, não OLTP: consultas ad-hoc sobre volumes históricos, para as quais arquiteturas lakehouse baseadas em armazenamento de objetos são a escolha adequada (ARMBRUST et al., 2021).

3.8.3 Limitações conhecidas da arquitetura proposta

As limitações descritas a seguir foram identificadas durante o desenvolvimento e a operação do sistema. Não invalidam os objetivos propostos, mas delimitam o escopo de aplicabilidade das conclusões e são retomadas com maior profundidade no Capítulo de Conclusão.

Ausência de reprocessamento automático em caso de falha. O pipeline de ETL não implementa mecanismo de retry automático para falhas da função Lambda ou do workflow de persistência no n8n. O dado de origem permanece intacto no Data Lake, permitindo reprocessamento manual, mas sem alerta automático em caso de lacuna analítica.

Latência introduzida pela memória de sessão via Redis. A leitura do histórico de sessão no Redis a cada invocação introduz dependência de um serviço externo ao fluxo principal. Uma indisponibilidade do Redis degrada a qualidade das respostas para interações que dependem de contexto acumulado, sem fallback implementado na versão atual.

Overhead da orquestração visual do n8n. A serialização e o agendamento de nós pelo motor interno do n8n introduzem latência adicional por nó em comparação a implementações equivalentes em Python puro. Para o escopo atual, esse overhead é secundário frente à latência dos modelos de linguagem, mas pode se tornar relevante em cenários de alta concorrência.

Dependência da consistência do JSON gerado pelos agentes. A qualidade das análises depende da conformidade estrutural dos arquivos JSON gravados no Data Lake. Modelos de linguagem não oferecem garantias formais de schema: em casos de instabilidade ou atualização de versão do LLM, campos esperados podem ser omitidos ou mal formatados, enviesando análises que dependam desses campos.

4 RESULTADOS

4.1 Contexto da Implantação

A Blanvur Dental Studio é uma clínica odontológica de alto padrão localizada em Curitiba (PR), com posicionamento *boutique* e foco em estética dental — em especial nas facetas híbridas, técnica que integra escaneamento digital, planejamento estético em software e impressão 3D. A clínica conta com laboratório digital próprio, dotado de tecnologia CAD/CAM para a produção interna de facetas e próteses, o que aproxima, em um mesmo ambiente, os fluxos de atendimento clínico, administrativo e protético-laboratorial que o sistema multiagente foi concebido para apoiar.

A validação empírica foi conduzida nesse ambiente real de produção. O período de coleta compreende o intervalo entre 3 e 25 de junho de 2026, correspondente às primeiras semanas de operação contínua após a implantação, durante o qual foram registradas 46 interações completas e rastreáveis.

O WhatsApp constituiu o único canal ativo no período, refletindo a realidade operacional da clínica, em que o aplicativo concentra a comunicação da equipe. A cada mensagem, o sistema classifica automaticamente sua origem em um dentre quatro perfis internos, distribuídos conforme a Tabela 4.1.

Tabela 4.1 — Distribuição das interações por perfil identificado

Perfil identificado	Interações
Dentista	29
Secretária	11
Estagiário	4
Técnico de laboratório	2
Total	46

Fonte: Elaborado pelo autor, a partir de consulta à tabela messages.

4.2 Funcionalidade do Sistema

4.2.1 Volume de uso e distribuição temporal

A continuidade operacional do sistema foi verificada por meio da contagem de interações agrupadas por dia, obtida diretamente da tabela messages para o período posterior ao início da operação (03/06/2026). A Tabela 4.2 apresenta a distribuição diária, acompanhada do dia da semana correspondente.

Tabela 4.2 — Distribuição diária das interações no período de coleta

Data (jun/2026)	Dia da semana	Interações
03	Quarta-feira	5
05	Sexta-feira	6
11	Quinta-feira	1
16	Terça-feira	3
17	Quarta-feira	4
18	Quinta-feira	5
19	Sexta-feira	5
20	Sábado	2
21	Domingo	5
22	Segunda-feira	3
23	Terça-feira	5
25	Quinta-feira	2

Total		46
-------	--	----

Fonte: Elaborado pelo autor, a partir de consulta à tabela messages.

Os dados confirmam que o sistema esteve em uso recorrente ao longo de todo o intervalo de coleta, com atividade registrada em doze dias distintos distribuídos entre 3 e 25 de junho de 2026. O volume diário oscilou entre 1 e 6 interações, com média aproximada de 3,8 interações por dia ativo. A maior parte da demanda concentrou-se em dias úteis, 39 das 46 interações (cerca de 85%) ocorreram de segunda a sexta-feira, padrão coerente com a operação administrativa e clínica de uma clínica odontológica, ainda que se observe atividade residual nos fins de semana.

Mais relevante para as decisões de arquitetura, contudo, é o caráter intermitente e de baixo volume do uso: a operação alterna períodos de atividade com dias inteiros ociosos, e mesmo nos dias mais movimentados o número de requisições é modesto. Esse perfil de carga, esparso, com baixo ciclo de trabalho e picos pontuais, constitui justamente o cenário em que a abordagem *serverless* adotada na arquitetura (Capítulo 3) é economicamente vantajosa: como a cobrança ocorre por execução e os recursos escalam a zero na ausência de demanda, não há custo de ociosidade, ao contrário do que ocorreria com uma infraestrutura provisionada de forma permanente. O padrão de uso observado, portanto, oferece sustentação empírica direta à escolha de projeto, e não apenas uma justificativa teórica.

4.2.2 Latência por tipo de input: texto versus multimodal

A funcionalidade de um assistente conversacional depende não apenas de responder corretamente, mas de fazê-lo em tempo aceitável. Para quantificar o desempenho percebido, calculou-se a latência de cada interação como a diferença entre `datetime_response` e `datetime_send`, segmentando os resultados pelo campo booleano `is_image_question`. Essa segmentação isola o custo temporal da rota multimodal, na qual a presença de uma imagem aciona condicionalmente o Agente A antes do encaminhamento ao especialista. A Tabela 4.3 sintetiza os resultados.

Tabela 4.3 — Latência de resposta por tipo de entrada

Tipo de entrada	Interações	Latência média (s)	Mínima (s)	Máxima (s)
Texto	40	22,1	2,8	100,5
Multimodal (imagem)	6	59,8	14,9	109,4

Fonte: Elaborado pelo autor, a partir de consulta à tabela messages.

Os resultados evidenciam um custo temporal expressivo associado ao processamento multimodal: a latência média das interações com imagem (59,8 s) é cerca de **2,7 vezes superior** à das interações exclusivamente textuais (22,1 s). Essa diferença é coerente com a arquitetura descrita no Capítulo 3, em que a entrada visual percorre uma etapa adicional — o roteamento e a descrição realizados pelo Agente A — antes de seguir para o agente especialista responsável pela resposta. O custo da capacidade multimodal, portanto, não é abstrato: manifesta-se de forma mensurável no tempo de resposta percebido pelo usuário.

Cabe ressaltar que ambas as faixas permanecem dentro de limites compatíveis com o canal utilizado. Por se tratar de comunicação assíncrona via WhatsApp, na qual a equipe não aguarda a resposta em tempo síncrono, uma latência média da ordem de dezenas de segundos não compromete a usabilidade. A elevada dispersão observada (mínimas de poucos segundos frente a máximas próximas de 100 s em ambas as categorias) reflete a variabilidade da complexidade das demandas e dos modelos de linguagem acionados em cada rota, tema retomado na análise de acionamento por agente (Seção 4.3.1).

4.3 Observabilidade do Sistema

A proposição de observabilidade desta tese sustenta que todo estado interno do sistema — do plano de roteamento à avaliação de qualidade de cada resposta — é consultável via SQL, sem acesso ao código-fonte ou à infraestrutura de execução. As subseções a seguir demonstram empiricamente essa propriedade a partir das tabelas `agents_plan` e `agents_actions`, derivadas pelo pipeline de ETL da Seção 3.6. Todas as análises consideram as 46 interações da janela operacional oficial (3 a 25 de junho de 2026), conforme a Seção 4.1.

4.3.1 Taxa de acionamento por agente especialista

A tabela `agents_plan` registra, para cada interação, os agentes incluídos no plano do Orquestrador (campo `agent_X_active`). A distribuição sobre as 46 interações é apresentada na Tabela 4.4.

Tabela 4.4 — Taxa de acionamento por agente especialista (n = 46 interações)

Agente	Domínio de especialização	Acionamentos	Taxa
A	Roteamento multimodal (imagens)	5	10,9%
B	Análise clínica e exames	10	21,7%
C	Leitura de rótulos e prints	2	4,3%
D	Atendimento comercial	14	30,4%
E	Atendimento ao paciente	16	34,8%
F	Suporte técnico de laboratório	3	6,5%
G	Geração de termos de consentimento	2	4,3%

Fonte: Elaborado pelo autor, a partir de consulta à tabela `agents_plan`.

O padrão reflete o perfil de uso da Seção 4.1: os Agentes E e D — atendimento ao paciente e abordagens comerciais — concentram as maiores frequências, compatíveis com o predomínio dos perfis Dentista e Secretária. O Agente B (clínico) ocupa a terceira posição com 21,7%, evidenciando peso relevante das demandas assistenciais, coerente com o foco da clínica em diagnóstico e planejamento de casos. Os Agentes C, F e G, de domínios específicos, apresentam taxas inferiores a 7% — casos de uso pontuais, mas rastreáveis com a mesma precisão. Essa distribuição seria inacessível sem a persistência explícita do plano de roteamento como registro consultável.

4.3.2 Análise de divergência entre plano e execução

A separação entre `agents_plan` (intenção declarada) e `agents_actions` (execução efetiva) permite identificar interações em que o Orquestrador planejou um percurso diferente do que o pipeline executou. A junção usa a chave composta `session_id + datetime_send`, e não apenas o `session_id`: este corresponde ao telefone do remetente no WhatsApp e repete-se entre interações do mesmo usuário; somente a combinação com o timestamp identifica univocamente cada interação.

A junção produziu correspondência integral entre as tabelas — as 46 interações presentes em ambas, sem órfãos. Dentro desse universo, **10 interações (21,7%) apresentaram divergência**, todas do mesmo tipo: o Orquestrador incluiu no plano um worker (Agente D ou E), mas a execução registrada em `agents_actions` não contém qualquer ação de agente (Tabela 4.5).

Tabela 4.5 — Interações com divergência entre plano e execução (n = 46 interações)

Especialista planejado	Ocorrências	Execução registrada
Agente E (atendimento)	6	Nenhum agente
Agente D (comercial)	4	Nenhum agente
Total	10	—

Fonte: Elaborado pelo autor, a partir da junção de `agents_plan` e `agents_actions` por `session_id + datetime_send`.

O cruzamento com a tabela `messages` mostra que boa parte dessas interações corresponde a turnos curtos de seguimento — confirmações breves, perguntas encadeadas a um tópico anterior, cumprimentos e meta-perguntas de baixa complexidade. A causa precisa não pode ser determinada a partir dos dados, pois o sistema não registra o evento que levaria o Orquestrador a não efetivar a delegação. Uma possibilidade a considerar, entre outras, relaciona-se à memória conversacional de curto prazo em Redis (janela de 15 turnos, TTL de 3600 s, Seção 3.4): com o contexto recente disponível em cache, o Orquestrador poderia

consolidar a resposta a partir do histórico retido, sem reefetivar a delegação. Trata-se, porém, de hipótese plausível, não confirmada. O fato de as divergências distribuírem-se por três remetentes distintos sugere comportamento recorrente, e não artefato pontual.

Esse comportamento é **parcialmente indesejado sob a perspectiva arquitetural**: o desenho do sistema pressupõe que cada demanda seja roteada ao especialista, de modo que a resposta incorpore o raciocínio especializado e seja submetida à auditoria do Agente H. Quando o Orquestrador responde diretamente, esse percurso é contornado, e a interação deixa de passar pelas garantias de especialização e verificação previstas. A investigação das causas e sua eventual correção constituem limitação identificada, remetida aos trabalhos futuros (Capítulo 5). Para a tese, o achado é duplamente relevante: o descompasso entre intenção declarada e execução efetiva é justamente o tipo de inconsistência invisível em arquiteturas monolíticas, e foi a própria observabilidade — uma junção SQL entre duas tabelas — que o tornou detectável e mensurável.

4.3.3 Avaliação qualitativa via Agente H

O Agente H opera como auditor interno: previsto para ser acionado ao final de cada pipeline, recebe a mensagem original e a resposta consolidada e produz uma avaliação estruturada persistida no campo `agent_h_actions_observation` da tabela `agents_actions` (Seção 3.4.5), em objeto JSON consultável com pontuações, checklist de alucinação e campos de revisão.

Cobertura efetiva da auditoria. Das 46 interações, 10 não tiveram execução de especialista (as divergências da Seção 4.3.2) e, portanto, nada havia a auditar. Das 36 restantes, o Agente H registrou avaliação em 33 — **cobertura efetiva de 91,7%**. Em três interações, apesar de o plano prever o Agente H e de um especialista ter respondido, nenhuma avaliação foi registrada. O resultado qualifica empiricamente a descrição de design da Seção 3.4.5: na operação real, o disparo do auditor, embora majoritário, não ocorreu na totalidade das interações elegíveis — lacuna que só a própria observabilidade torna detectável.

Pontuações por dimensão. Dos 32 registros com avaliação, 25 continham pontuações numéricas; os 7 restantes correspondem a casos em que o Agente H foi acionado sem resposta consolidada para auditar, solicitando o conteúdo em vez de pontuar. As médias derivam dos 25 registros pontuados (Tabela 4.6).

Tabela 4.6 — Pontuações médias por dimensão avaliada pelo Agente H (n = 25)

Dimensão	Média	Mínimo	Máximo
Limites respeitados	7,20	0	10
Clareza	6,32	0	10
Suporte contextual	5,76	0	10
Precisão técnica	5,64	0	10
Fonte identificada	4,04	0	10

Fonte: Elaborado pelo autor, a partir do campo `agent_h_actions_observation` da tabela `agents_actions`.

A maior média é `limites_respeitados` (7,20), indicando que o sistema evita comprometimentos indevidos — diagnósticos definitivos ou promessas de resultado — na maior parte das interações, em conformidade com os limites dos `system prompts`. A mais frágil é `fonte_identificada` (4,04): as respostas frequentemente não distinguem estimativa técnica de dado verificável, comportamento esperado em LLMs generalistas e que sustenta a necessidade da auditoria.

Checklist de alucinação e revisão. O flag `afirmacao_sem_base` ocorreu em 8 interações (25,0%), concentradas em respostas dos Agentes B e F com parâmetros numéricos específicos (doses de anestésico, tempos de pós-cura) apresentados sem condicionar à ficha técnica ou ao protocolo local. Os flags `contradicao` e `promessa_resultado` não registraram qualquer ocorrência — a ausência deste último é relevante no contexto odontológico, onde garantias de resultado são vedadas. Por fim, em 28 dos 32 registros o Agente H preencheu o campo `motivo_revisao` com indicação de ajustes antes do envio. O acionamento *automatizado* de revisão humana a partir dessa sinalização está previsto como evolução futura e não integra o escopo deste trabalho (Seção 3.4.5); o campo permanece registrado e consultável via SQL, base direta para essa extensão — o que reforça, mais uma vez, a observabilidade defendida pela tese.

4.4 Viabilidade Econômica

As proposições de funcionalidade e observabilidade só sustentam a tese se acompanhadas de uma terceira: a de que o sistema é economicamente viável para organizações de pequeno e médio porte. Esta seção quantifica o custo real de operação no período de implantação (3 a 25 de junho de 2026), separando a despesa de inferência da despesa da camada analítica, e o confronta com o conjunto de competências entregue à organização. Os valores de inferência derivam do painel do OpenRouter e os de infraestrutura, do AWS Cost Explorer; nenhum integra a lógica dos agentes, constituindo medição posterior à operação.

4.4.1 Custo total de inferência no período

O OpenRouter contabiliza o custo por chamada a modelo de linguagem, não por interação ou sessão. O custo do período foi, portanto, obtido pela soma de todas as chamadas, e o custo médio por interação, pela razão entre esse total e as 46 interações da tabela messages. No intervalo, as 46 interações originaram **165 chamadas a modelos** ($\approx 3,6$ por interação, compatível com o encadeamento Orquestrador $\rightarrow$ especialista $\rightarrow$ Agente H), somando **US\$ 2,52**, distribuídos conforme a Tabela 4.7.

Tabela 4.7 — Custo de inferência por modelo no período (3 a 25 de junho de 2026)

Modelo (papel na arquitetura)	Custo (US\$)	Participação
Gemini 3 Flash (Orquestrador e Agente B)	2,2682	90,0%
GPT Nano (Agentes D, E e H)	0,1824	7,2%
Qwen 3.6 Plus (Agentes F e G)	0,0698	2,8%
Gemma 4 26B (Agentes A e C)	0,0011	0,04%
Total	2,5215	100%

Fonte: Elaborado pelo autor, a partir do painel de atividade do OpenRouter.

O custo total é diminuto: pouco mais de dois dólares e meio para vinte e três dias de operação, ou **US\$ 0,055 por interação** — cerca de **R\$ 0,28**, ao câmbio de R\$ 5,17/US\$ vigente ao final do período. A despesa concentra-se no Gemini 3 Flash (90,0%), que atua como Orquestrador em toda interação; a concentração não decorre do preço unitário, mas do volume de tokens que o Orquestrador processa, frequentemente acima de 100 mil por chamada ao carregar o contexto de roteamento, ante prompts muito menores dos especialistas. A própria observabilidade de custo identifica, assim, o componente economicamente dominante.

4.4.2 Custo da camada analítica (Athena)

A observabilidade defendida pela tese repousa sobre o pipeline serverless da Seção 3.6 (S3 → Lambda → Athena). A exportação do AWS Cost Explorer para o período registra despesa **efetivamente nula** em todos esses serviços — frações desprezíveis de centavo, abaixo do limiar de cobrança. O resultado é estrutural: o Athena cobra por volume escaneado (poucos dólares por terabyte), e os logs do período somam apenas alguns megabytes, de modo que cada consulta varre fração ínfima dessa unidade. A consequência é direta: no modelo adotado, a observabilidade tem **custo marginal** e pode ser exercida intensivamente — como nas Seções 4.3.1 a 4.3.3 — sem impacto orçamentário perceptível.

4.4.3 Síntese de viabilidade para PMEs

A R\$ 0,28 por interação, o custo automatizado já é favorável frente ao trabalho humano mesmo na comparação mais conservadora com o salário mínimo de 2026 (R\$ 1.621,00; custo ao empregador da ordem de R\$ 2.700,00 mensais). Essa comparação, porém, subestima o que é entregue, pois pressupõe um único trabalhador genérico. O parâmetro adequado é a **multi-capacidade do sistema**: as 46 interações mobilizaram sete domínios distintos de competência (Seção 4.3.1) — análise clínica, atendimento comercial, atendimento ao paciente, suporte de laboratório, geração de termos de consentimento, leitura de imagens e roteamento. Em uma estrutura humana, essas funções não caberiam em um cargo de piso: a redação de um termo de consentimento exige conhecimento normativo, o suporte de laboratório exige competência técnica e a triagem clínica exige formação na área — profissionais que uma clínica de pequeno porte não manteria simultaneamente em folha.

O parâmetro decisivo, portanto, não é o custo por pessoa, mas o **custo total frente ao que a operação entrega**. Projetada para o volume observado, a despesa mensal somada de

inferência e infraestrutura permanece na ordem de poucas dezenas de reais — inferior a uma única hora de trabalho qualificado — e por ela a clínica obtém atendimento multi-domínio contínuo, integrado ao WhatsApp e rastreável via SQL. Esse patamar é viabilizado pela ausência de custo fixo do modelo serverless, que escala a zero na inatividade (Seção 4.2.1) e elimina a despesa de ociosidade incompatível com o uso intermitente da clínica. Cai, com isso, a premissa de que IA em operação real seria viável apenas para grandes organizações: o pagamento por uso aproxima essa capacidade do orçamento de uma pequena empresa, deslocando a barreira de adoção do custo para o conhecimento técnico de implantação.

Cabe registrar os limites: os valores refletem o baixo volume observado e a seleção de modelos, são sensíveis à variação cambial e a preços dos provedores, e não incorporam custos de desenvolvimento e manutenção (Capítulo 5). Dentro dessas fronteiras, os dados sustentam a terceira proposição da tese — o sistema entrega, a um custo total compatível com uma organização de pequeno porte, um conjunto de competências que de outro modo lhe seria inacessível.

4.5 Síntese dos Resultados

Os resultados apresentados nas seções anteriores podem ser organizados em torno das três proposições enunciadas na tese. O Quadro 4.1 sintetiza, para cada uma, a evidência empírica que a sustenta, a seção de origem e as limitações identificadas cuja discussão é desenvolvida no Capítulo 5.

Quadro 4.1 — Proposições da tese: evidências, seções de origem e limitações identificadas

Proposição	Evidência empírica	Seção	Limitações identificadas
Funcionalidade	46 interações em 23 dias, quatro perfis, sete domínios de competência; latência média de 22,1 s (texto) e 59,8 s (multimodal), compatíveis com o canal assíncrono	4.1, 4.2.1, 4.2.2	Volume reduzido; janela curta; ausência de avaliação de satisfação

Observabilidade	Todas as análises derivadas de SQL sobre três tabelas do DW, sem acesso a código-fonte ou infraestrutura; 10 divergências plano—execução (21,7%) detectadas via junção; cobertura do Agente H em 91,7% das interações elegíveis	4.3.1, 4.3.2, 4.3.3	Causa das divergências não determinável pelos dados persistidos; 3 interações elegíveis sem auditoria; revisão humana automatizada não implementada
Viabilidade econômica	Custo de inferência de US\$ 2,52 (R\$ 0,28/interação); camada analítica com custo efetivamente nulo; modelo serverless com custo zero em ociosidade	4.4.1, 4.4.2, 4.4.3	Valores condicionados ao volume e à seleção de modelos; sensibilidade cambial; custos de desenvolvimento e manutenção não incorporados

Fonte: Elaborado pelo autor.

A funcionalidade é sustentada pelo registro de uso efetivo em produção: 46 interações originadas por quatro perfis profissionais, cobrindo sete domínios — do roteamento multimodal à geração de termos de consentimento —, com latência compatível com o canal assíncrono. O volume reduzido e a janela curta, contudo, limitam a generalização e são discutidos no Capítulo 5.

A observabilidade é sustentada não apenas por achados pontuais, mas pelo próprio método de obtenção de todos os resultados deste capítulo: cada tabela, percentual e divergência foram derivados de consultas SQL, sem acesso ao código-fonte ou à infraestrutura. As lacunas — divergências de causa indeterminada, interações sem auditoria — são igualmente reportadas com transparência, reforçando o argumento: a observabilidade permite identificar tanto o que o sistema fez quanto o que deixou de fazer.

A viabilidade econômica é sustentada pela quantificação do custo real: R\$ 0,28 por interação e custo analítico efetivamente nulo, resultando em despesa mensal inferior a uma hora

de trabalho qualificado para sete domínios de competência. O modelo serverless elimina a ociosidade que inviabilizaria a adoção por uma clínica com perfil de uso intermitente.

As três proposições se sustentam mutuamente: a funcionalidade sem observabilidade seria uma caixa-preta; a observabilidade sem viabilidade econômica ficaria restrita a organizações com orçamento de P&D dedicado; e a viabilidade sem funcionalidade demonstrada seria uma promessa sem lastro empírico. Os limites identificados não invalidam as proposições dentro do escopo da implantação, mas circunscrevem o alcance das conclusões e orientam os trabalhos futuros desenvolvidos no Capítulo 5.

5 CONCLUSÃO

Este trabalho partiu de três gargalos recorrentes na adoção de inteligência artificial por organizações fora do eixo das grandes empresas de tecnologia: a inflexibilidade dos chatbots monolíticos diante de tarefas que exigem especialização; o acoplamento entre inferência e persistência, fonte de latência e de perda de dados; e a opacidade da caixa-preta, que impede auditar o raciocínio da IA e mensurar seu custo e desempenho de forma estruturada. A tese defendida foi a de que uma arquitetura serverless orientada a eventos pode enfrentar os três simultaneamente, sustentando um sistema multiagente com rastreabilidade total das interações e mensuração objetiva de custo, desempenho e qualidade — condição para viabilizar a adoção de IA conversacional em pequenas e médias organizações.

Desse enunciado derivou o objetivo geral: projetar, implantar e avaliar empiricamente, em uma clínica odontológica de pequeno porte, uma arquitetura serverless orientada a eventos para um sistema multiagente de IA conversacional, capaz de automatizar tarefas especializadas com rastreabilidade total e custo mensurável. Esse objetivo desdobrou-se em quatro objetivos específicos, cuja verificação organiza a síntese a seguir.

O primeiro objetivo específico — implementar o ecossistema multiagente (Orquestrador e oito agentes especialistas), com roteamento multimodal, memória de sessão e loop de auto-avaliação — foi atingido e evidenciado pelo uso efetivo em produção: as 46 interações registradas em 23 dias, originadas por quatro perfis profissionais e distribuídas por sete domínios de competência (Seções 4.1 e 4.2), demonstram que o sistema atendeu demandas reais e heterogêneas, da triagem clínica à geração de termos de consentimento, com latência compatível com o canal assíncrono (22,1 s em média para texto e 59,8 s para entradas multimodais).

O segundo objetivo específico — construir o pipeline de dados orientado a eventos (S3 → Lambda → Athena) que persiste e normaliza cada interação em um data warehouse consultável — foi cumprido e é, ele próprio, a condição de possibilidade de todo o Capítulo 4: cada tabela, percentual e divergência ali apresentados derivou de consultas SQL sobre três tabelas do data warehouse, sem qualquer acesso ao código-fonte ou à infraestrutura de execução (Seção 4.3).

O terceiro objetivo específico — demonstrar a rastreabilidade total do comportamento do sistema — sustentou-se pelos achados que só a observabilidade torna detectáveis: as 10 divergências entre plano e execução (21,7% das interações), identificadas por junção entre as tabelas `agents\_plan` e `agents\_actions`, e a cobertura de 91,7% do Agente H sobre as interações elegíveis (Seções 4.3.2 e 4.3.3). Reportar com transparência não apenas o que o sistema fez, mas também o que deixou de fazer, é o que confere força à proposição: a arquitetura permite auditar tanto o acerto quanto a lacuna.

O quarto objetivo específico — mensurar objetivamente custo, desempenho e qualidade — concretizou-se na quantificação do custo real de operação: US\$ 2,52 em 23 dias, ou R\$ 0,28 por interação, com custo efetivamente nulo na camada analítica e ausência do custo fixo de ociosidade próprio de servidores dedicados, característica do modelo serverless (Seção 4.4). A qualidade, por sua vez, foi objetivada pelas pontuações do Agente H e pelo checklist de alucinação (Seção 4.3.3). Verificados os quatro objetivos específicos, o objetivo geral considera-se alcançado: as três proposições da tese — funcionalidade, observabilidade e viabilidade — sustentaram-se dentro do escopo da implantação, e o fizeram de forma interdependente, pois a funcionalidade sem observabilidade seria uma caixa-preta, a observabilidade sem viabilidade ficaria restrita a orçamentos de P&D, e a viabilidade sem funcionalidade seria promessa sem lastro empírico.

A principal contribuição do trabalho situa-se em uma lacuna apontada pela literatura. Khanagar et al. (2021), ao revisarem a produção sobre IA na odontologia, encontraram concentração quase integral em diagnóstico por imagem, sem sistemas conversacionais, automação administrativa ou geração de documentação no domínio. Este trabalho preenche parte dessa lacuna ao demonstrar, em operação real, um sistema que cobre simultaneamente domínios clínicos, comerciais, laboratoriais e jurídicos, com rastreabilidade completa. A contribuição, porém, é menos sobre a odontologia e mais sobre um método: tratar a observabilidade não como recurso de depuração, mas como camada de mensuração de baixo custo que torna o comportamento de um sistema de IA auditável via SQL — deslocando a barreira de adoção de IA em pequenas e médias organizações do custo de infraestrutura para o conhecimento técnico de implantação.

Os resultados devem ser lidos dentro de fronteiras claras. O volume de 46 interações e a janela de 23 dias limitam a generalização e não comportam inferência estatística, e não houve

avaliação de satisfação dos usuários. A causa das divergências entre plano e execução não é determinável a partir dos dados persistidos — a hipótese do comportamento do cache Redis, levantada na Seção 4.3, permanece não confirmada. O disparo do Agente H, embora majoritário, não foi universal, e o acionamento automatizado de revisão humana não chegou a ser implementado. Por fim, os valores de custo estão condicionados ao baixo volume, à seleção de modelos e à variação cambial, e não incorporam custos de desenvolvimento e manutenção.

Essas mesmas fronteiras delineiam os trabalhos futuros. No plano da inteligência dos agentes, apontam-se o ajuste fino (fine-tuning) para o domínio odontológico, a ampliação do uso de RAG e a integração a ferramentas e fontes de dados externas, e a extensão do roteamento multimodal para a ingestão de documentos em PDF. No plano da observabilidade, destacam-se um painel de BI que consolide as três tabelas hoje analisadas de forma independente e o aprofundamento analítico pela avaliação individualizada das respostas de cada agente e pela análise sistemática das perguntas recebidas. Somam-se a implementação do acionamento automatizado de revisão humana a partir dos flags já persistidos e a validação em maior escala e janela mais longa.

Dentro do escopo delimitado, os dados coletados sustentam a tese proposta: uma arquitetura serverless orientada a eventos mostrou-se capaz de sustentar um sistema multiagente funcional, observável e economicamente viável, aproximando a adoção de IA conversacional rastreável da realidade orçamentária de uma organização de pequeno porte.

REFERÊNCIAS

AMAZON WEB SERVICES. Amazon Athena pricing. Seattle: AWS, 2024. Disponível em: <https://aws.amazon.com/athena/pricing/>. Acesso em: mai. 2025.

ARMBRUST, Michael et al. Lakehouse: A New Generation of Open Platforms that Unify Data Warehousing and Advanced Analytics. In: CONFERENCE ON INNOVATIVE DATA SYSTEMS RESEARCH (CIDR), 11., 2021. Proceedings... [S.l.]: CIDR, 2021. Disponível em: https://www.cidrdb.org/cidr2021/papers/cidr2021_paper17.pdf. Acesso em: abr. 2025.

BOMMASANI, Rishi et al. On the Opportunities and Risks of Foundation Models. arXiv, 2021.

BRASIL. Lei nº 13.709, de 14 de agosto de 2018. Lei Geral de Proteção de Dados Pessoais (LGPD). Disponível em: https://www.planalto.gov.br/ccivil_03/_ato2015-2018/2018/lei/l13709.htm.

BROWN, Tom et al. Language Models are Few-Shot Learners. In: ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS, 2020.

DECAN, Alexandre; MENS, Tom; GROSJEAN, Philippe. An Empirical Comparison of Dependency Network Evolution in Seven Software Packaging Ecosystems. Empirical Software Engineering, v. 24, n. 1, p. 381–416, 2018. Disponível em: <https://arxiv.org/abs/1710.04669>. Acesso em: abr. 2025.

DIJKSTRA, Edsger W. On the Role of Scientific Thought. EWD447. Nuenen, 1974. Disponível em: <https://www.cs.utexas.edu/users/EWD/ewd04xx/EWD447.PDF>. Acesso em: abr. 2025.

IETF — INTERNET ENGINEERING TASK FORCE. RFC 6750: The OAuth 2.0 Authorization Framework: Bearer Token Usage. Fremont: IETF, 2012. Disponível em: <https://datatracker.ietf.org/doc/html/rfc6750>. Acesso em: abr. 2025.

JONAS, Eric et al. Cloud Programming Simplified: A Berkeley View on Serverless Computing. arXiv, 2019.

KHANAGAR, Sanjeev B. et al. Developments, application, and performance of artificial intelligence in dentistry: a systematic review. *Journal of Dental Sciences*, v. 16, n. 1, p. 508–522, 2021. DOI: 10.1016/j.jds.2020.06.019.

KLEPPMANN, Martin. *Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems*. Sebastopol: O'Reilly Media, 2017.

KUMAR, Arun. How to Connect AI Agents with n8n for End-to-End Automation. *Global Knowledge Academy*, v. 5, n. 4, p. 73–81, 2024. Disponível em: <https://globalknowledgeacademy.com/index.php/gna/article/view/76>. Acesso em: abr. 2025.

LANGGRAPH DOCUMENTATION. *LangGraph: Build stateful, multi-actor applications with LLMs*. LangChain, 2024. Disponível em: <https://langchain-ai.github.io/langgraph/>. Acesso em: abr. 2025.

LEE, Peter et al. Benefits, limits, and risks of GPT-4 as an AI chatbot for medicine. *New England Journal of Medicine*, v. 388, n. 13, p. 1233–1239, 2023. DOI: 10.1056/NEJMSr2214184.

MICHELSON, Brenda M. *Event-Driven Architecture Overview: Event-Driven SOA Is Just Part of the EDA Story*. Boston: Patricia Seybold Group, 2006. DOI: 10.1571/bda2-2-06cc.

NARGESIAN, Fatemeh et al. Data Lake Management: Challenges and Opportunities. *Proceedings of the VLDB Endowment*, v. 12, n. 12, p. 1986–1989, 2019. Disponível em: <https://dl.acm.org/doi/10.14778/3352063.3352116>. Acesso em: mai. 2025.

OPENAI. GPT-4 Technical Report. arXiv, 2023.

OUYANG, Long et al. Training Language Models to Follow Instructions with Human Feedback. In: *ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS*, 2022.

OWASP — OPEN WORLDWIDE APPLICATION SECURITY PROJECT. OWASP API Security Top 10. [S.l.]: OWASP Foundation, 2023. Disponível em: <https://owasp.org/API-Security>. Acesso em: abr. 2025.

RADFORD, Alec et al. Learning Transferable Visual Models From Natural Language Supervision. In: *INTERNATIONAL CONFERENCE ON MACHINE LEARNING*, 2021.

SHINN, Noah; CASSANO, Federico; LABASH, Ashwin; GOPALAN, Karthik; NARASIMHAN, Karthik; YAO, Shunyu. Reflexion: Language Agents with Verbal Reinforcement Learning. In: ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS, 36., 2023, New Orleans. Proceedings... Red Hook: Curran Associates, 2023.

Disponível em:

https://proceedings.neurips.cc/paper_files/paper/2023/file/1b44b878bb782e6954cd888628510e90-Paper-Conference.pdf. Acesso em: mai. 2025.

THUSOO, Ashish et al. Hive: a warehousing solution over a map-reduce framework. Proceedings of the VLDB Endowment, v. 2, n. 2, p. 1626–1629, 2009. Disponível em: <https://dl.acm.org/doi/10.14778/1687553.1687609>. Acesso em: mai. 2025.

VASWANI, Ashish et al. Attention Is All You Need. In: ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS, 2017.

WANG, Lei et al. A Survey on Large Language Model based Autonomous Agents. Frontiers of Computer Science, 2024.

WANG, Liang et al. Peeking Behind the Curtains of Serverless Platforms. In: USENIX ANNUAL TECHNICAL CONFERENCE (ATC), 2018, Boston. Proceedings... Berkeley: USENIX, 2018. p. 133–146. Disponível em: <https://www.usenix.org/system/files/conference/atc18/atc18-wang-liang.pdf>. Acesso em: mai. 2025.

WEI, Jason et al. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In: ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS, 2022.

WOOLDRIDGE, Michael; JENNINGS, Nicholas R. Intelligent Agents: Theory and Practice. The Knowledge Engineering Review, Cambridge, v. 10, n. 2, p. 115–152, 1995.

WU, Qingyun et al. AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation Framework. arXiv, 2023. Disponível em: <https://arxiv.org/abs/2308.08155>. Acesso em: abr. 2025.

YAO, Shunyu et al. ReAct: Synergizing Reasoning and Acting in Language Models. In: INTERNATIONAL CONFERENCE ON LEARNING REPRESENTATIONS (ICLR), 2023,

Kigali. Proceedings... [S.l.]: OpenReview, 2023. Disponível em:
<https://arxiv.org/pdf/2210.03629>. Acesso em: abr. 2025.

ZHANG, Zeyu et al. A Survey on the Memory Mechanism of Large Language Model based Agents. arXiv, 2024.