



UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE ENGENHARIA DE AUTOMAÇÃO E SISTEMAS
CURSO DE GRADUAÇÃO EM ENGENHARIA DE CONTROLE E AUTOMAÇÃO

Gabriel Machado Rosset

**Otimização de Sistemas de IA Generativa e RAG para Integração de Bases de
Dados de Saúde em Suporte a Vendas Complexas: Estudo de Arquiteturas para
a Plataforma de Inteligência LIA**

Florianópolis
2026

Gabriel Machado Rosset

Otimização de Sistemas de IA Generativa e RAG para Integração de Bases de Dados de Saúde em Suporte a Vendas Complexas: Estudo de Arquiteturas para a Plataforma de Inteligência LIA

Relatório final da disciplina DAS5511 (Projeto de Fim de Curso) como Trabalho de Conclusão do Curso de Graduação em Engenharia de Controle e Automação da Universidade Federal de Santa Catarina em Florianópolis.

Orientador: Prof. Carlos Barros Montez, Dr.

Supervisor: Marcia Cristina Rodrigues Medina Kormann Alexandre

Florianópolis

2026

Ficha catalográfica gerada por meio de sistema automatizado gerenciado pela BU/UFSC.
Dados inseridos pelo próprio autor.

Rosset, Gabriel Machado

Otimização de Sistemas de IA Generativa e RAG para Integração de Bases de Dados de Saúde em Suporte a Vendas Complexas: Estudo de Arquiteturas para a Plataforma de Inteligência LIA / Gabriel Machado Rosset ; orientador, Carlos Barros Montez, coorientadora, Marcia Cristina Rodrigues Medina Kormann Alexandre, 2026.

87 p.

Trabalho de Conclusão de Curso (graduação) -
Universidade Federal de Santa Catarina, Centro Tecnológico,
Graduação em Engenharia de Controle e Automação,
Florianópolis, 2026.

Inclui referências.

1. Engenharia de Controle e Automação. 2. LLM. 3. IA Agentes . 4. MCP. 5. Otimização. I. Montez, Carlos Barros . II. Alexandre, Marcia Cristina Rodrigues Medina Kormann . III. Universidade Federal de Santa Catarina. Graduação em Engenharia de Controle e Automação. IV. Título.

Gabriel Machado Rosset

Otimização de Sistemas de IA Generativa e RAG para Integração de Bases de Dados de Saúde em Suporte a Vendas Complexas: Estudo de Arquiteturas para a Plataforma de Inteligência LIA

Esta monografia foi julgada no contexto da disciplina DAS5511 (Projeto de Fim de Curso) e aprovada em sua forma final pelo Curso de Graduação em Engenharia de Controle e Automação

Florianópolis, 07 de julho de 2026.

Marcelo De Lellis Costa de Oliveira
Coordenador do Curso

Banca Examinadora:



Documento assinado digitalmente
Carlos Barros Montez
Data: 15/07/2026 15:38:34-0300
CPF: ***.035.027-**
Verifique as assinaturas em <https://v.ufsc.br>

Carlos Barros Montez

Orientador(a)



UFSC/CTC/FAS
Documento assinado digitalmente
MARCIA CRISTINA RODRIGUES ME...ALEXANDRE
Data: 17/07/2026 09:54:41-0300
CPF: ***.052.601-**
Verifique as assinaturas em <https://v.ufsc.br>

Marcia Medina Kormann Alexandre

Supervisor(a)

Lifeshub

Jomi Fred Hübner

Avaliador(a)

UFSC/CTC/EAS

Hector Bessa Silveira

Presidente da Banca

UFSC/CTC/EAS

AGRADECIMENTOS

Agradeço, antes de tudo, à minha família, por todo o apoio, suporte e carinho ao longo desta trajetória. Sou profundamente grato por serem como são e por terem me criado da melhor maneira possível, sempre com dedicação, presença e cuidado. É por eles que consegui chegar até aqui e concluir esta etapa da minha formação.

Aos meus amigos da faculdade, à família Automação e as entidades do curso que fizeram parte dessa caminhada e ajudaram muito para que esta etapa fosse concluída. Ninguém se forma sozinho, e o fato de o curso ter sido marcado por tanta união certamente fez diferença para que esse período fosse uma experiência boa e significativa até chegar a este trabalho.

À Lifeshub pelas oportunidades oferecidas, pela experiência prática e por me colocar em situações nas quais pude amadurecer e crescer profissionalmente. Agradeço especialmente ao Cairo, pela mentoria, pela confiança e por sempre ter me ajudado quando precisei.

Ao meu orientador Prof. Carlos Barros Montez, e à minha supervisora Marcia pelas dicas e conselhos nesse trabalho.

Por fim, aos grandes amigos e pessoas importantes que estão na minha vida desde antes da faculdade Pietra, Brenos, Murilo, Emanuel e Alexandre.

DECLARAÇÃO DE PUBLICIDADE

Florianópolis, 29 de junho de 2026.

Na condição de supervisora vinculada à Lifeshub, instituição na qual o presente trabalho foi realizado, declaro não haver ressalvas quanto ao aspecto de sigilo ou propriedade intelectual sobre as informações contidas neste documento, que impeçam a sua publicação por parte da Universidade Federal de Santa Catarina (UFSC) para acesso pelo público em geral, incluindo a sua disponibilização *online* no Repositório Institucional da Biblioteca Universitária da UFSC. Além disso, declaro ciência de que o autor, na condição de estudante da UFSC, é obrigado a depositar este documento, por se tratar de um Trabalho de Conclusão de Curso, no referido Repositório Institucional, em atendimento à Resolução Normativa n° 126/2019/CUn.

Por estar de acordo com esses termos, subscrevo-me abaixo.

Documento assinado digitalmente
 **MARCIA CRISTINA RODRIGUES MEDINA KORMAI**
Data: 15/07/2026 13:15:54-0300
Verifique em <https://validar.iti.gov.br>

Marcia Cristina Rodrigues Medina Kormann Alexandre
Lifeshub

RESUMO

Em um contexto de ampla adoção de LLMs, a inovação por uso dessa tecnologia deve mudar como trabalhamos em diversas áreas, porém as IAs ainda apresentam alguns problemas, e grande parte deles não são intrínsecos a essa tecnologia, mas sim a como a usamos, assim havendo espaço para a otimização de técnicas e arquiteturas de uso para chegar a um resultado melhor e mais eficiente. Este trabalho parte do contexto da plataforma conversacional LIA da Lifeshub voltada a apoiar vendas complexas no setor de saúde por meio da consulta integrada a bases cadastrais, empresariais, assistenciais e de relacionamento comercial. Em agentes de inteligência artificial generativa que usam ferramentas externas, esse tipo de consulta tende a elevar consumo de *tokens*, latência, custo e instabilidade, principalmente quando a pergunta exige cruzamento de entidades e preservação factual entre fontes. O objetivo é implementar e avaliar otimizações na camada de orquestração da *Lineage Intelligence Agent* (LIA), sem propor o desenvolvimento da plataforma em si, para tornar o uso de modelos de linguagem mais eficiente e confiável. As otimizações investigadas incluem seleção semântica de ferramentas, compactação de *payloads* e contexto, uso de memória operacional, compressão cognitiva, recuperação auxiliar de evidências e separação entre recuperação e composição da resposta. Essas estratégias buscam reduzir informação desnecessária enviada ao modelo, organizar melhor as evidências e diminuir decisões repetitivas ou frágeis do agente. A avaliação foi conduzida por um *benchmark* de ablação com diferentes arquiteturas e modelos, medindo acurácia, qualidade factual, latência, consumo de *tokens* e uso de ferramentas. Os resultados indicam que as otimizações podem melhorar custo e eficiência em parte dos cenários, mas que o melhor arranjo depende do modelo e do tipo de tarefa. O trabalho contribui com uma metodologia experimental para avaliar arquiteturas de agentes com ferramentas em aplicações corporativas de saúde.

Palavras-chave: Inteligência artificial generativa. Agentes com ferramentas. Orquestração de LLMs. RAG. Benchmark.

ABSTRACT

In a context of widespread adoption of LLMs, innovation through the use of this technology is expected to change how we work in several areas. However, AIs still present some problems, and most of them are not intrinsic to the technology itself, but rather to how we use it. Thus, there is room for optimizing techniques and usage architectures to achieve better and more efficient results. This work is set in the context of LIA, a Lifeshub conversational platform designed to support complex sales in the health sector by querying integrated healthcare, business, administrative, and commercial relationship data sources. In generative artificial intelligence agents that rely on external tools, this type of query tends to increase token consumption, latency, cost, and instability, especially when the task requires entity linking and factual preservation across sources. The objective is to implement and evaluate optimizations in LIA's orchestration layer, rather than developing the platform itself, to make the use of language models more efficient and reliable. The investigated optimizations include semantic tool selection, payload and context compaction, operational memory, cognitive compression, auxiliary evidence retrieval, and separation between retrieval and answer composition. These strategies aim to reduce unnecessary information sent to the model, organize evidence more effectively, and reduce repetitive or fragile agent decisions. The evaluation was conducted through an ablation benchmark with different architectures and models, measuring accuracy, factual quality, latency, token consumption, and tool usage. The results indicate that the optimizations can improve cost and efficiency in part of the scenarios, but the best arrangement depends on the model and task type. The work contributes an experimental methodology for evaluating tool-using agent architectures in corporate health applications.

Keywords: Generative artificial intelligence. Tool-using agents. LLM orchestration. RAG. Benchmark.

LISTA DE FIGURAS

Figura 1 – Fluxo lógico da solução avaliada	38
Figura 2 – Custo monetário estimado por acerto nas melhores configurações .	56
Figura 3 – Melhor resultado por modelo nas configurações avaliadas	59
Figura 4 – <i>Tokens</i> por acerto nas melhores configurações de cada modelo . .	59
Figura 5 – Fronteira entre acertos completos e custo monetário estimado por acerto	60
Figura 6 – Fronteira entre <i>score</i> factual e custo monetário estimado por acerto	61

LISTA DE QUADROS

Quadro 1 – Conceitos teóricos utilizados no trabalho	29
Quadro 2 – Grupos de ferramentas considerados no recorte experimental . . .	33
Quadro 3 – Etapas da linha de base V2 da LIA	39
Quadro 4 – Fronteira entre modelo, ferramenta e base de dados	40
Quadro 5 – Arquiteturas avaliadas e hipótese técnica	41
Quadro 6 – Perguntas e comportamentos esperados dos testes L1 a L5	49
Quadro 7 – Perguntas e comportamentos esperados dos testes L6 a D1	50
Quadro 8 – Critérios usados na leitura dos resultados	54
Quadro 9 – Custo técnico e custo monetário estimado nas melhores configura- ções	56
Quadro 10 – Porte, categoria de capacidade e preço por milhão de <i>tokens</i> dos modelos avaliados	58
Quadro 11 – Panorama inicial dos modelos e principais variantes	62
Quadro 12 – Arquiteturas com maior cobertura e seus resultados médios	63
Quadro 13 – Arquiteturas com melhor desempenho por cenário de teste	64
Quadro 14 – Efeito pareado dos componentes arquiteturais	65
Quadro 15 – Destaques de arquiteturas para Gemma4 31B	67
Quadro 16 – Destaques de arquiteturas para Nemotron 3 Super	68
Quadro 17 – Destaques de arquiteturas para GPT-OSS 20B	69
Quadro 18 – Destaques de arquiteturas para MiniMax M2.7	70
Quadro 19 – Destaques de arquiteturas para GPT-OSS 120B	71
Quadro 20 – Destaques de arquiteturas para DeepSeek V4 Flash	72
Quadro 21 – Destaques de arquiteturas para Kimi K2.6	73
Quadro 22 – Destaques de arquiteturas para MiniMax M3 na rodada completa .	74
Quadro 23 – Destaques de arquiteturas para MiniMax M3 na rodada parcial . .	74

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ACC	<i>Agent Cognitive Compressor</i>
AT	AutoTool
CNES	Cadastro Nacional de Estabelecimentos de Saúde
CP	<i>Compact Payload</i>
CRM	<i>Customer Relationship Management</i>
ETL	<i>Extract, Transform and Load</i>
IA	Inteligência Artificial
JSON	<i>JavaScript Object Notation</i>
LIA	<i>Lineage Intelligence Agent</i>
LLM	<i>Large Language Model</i>
MB	Membox
MCP	<i>Model Context Protocol</i>
QSA	Quadro de Sócios e Administradores
RAG	<i>Retrieval-Augmented Generation</i>
RTS	<i>Retrieve-then-Solve</i>
STR	<i>Semantic Tool Retrieval</i>
TR	TeaRAG

SUMÁRIO

1	INTRODUÇÃO	16
1.1	CONTEXTUALIZAÇÃO	16
1.2	DEFINIÇÃO DO PROBLEMA	17
1.3	OBJETIVOS	17
1.3.1	Objetivo Geral	17
1.3.2	Objetivos Específicos	17
1.4	ESCOPO E DELIMITAÇÕES	18
1.5	SOLUÇÃO PROPOSTA E METODOLOGIA GERAL	19
1.6	CONTRIBUIÇÕES DO TRABALHO	19
1.7	ESTRUTURA DO DOCUMENTO	19
2	FUNDAMENTAÇÃO TEÓRICA	21
2.1	MODELOS DE LINGUAGEM DE LARGA ESCALA	21
2.1.1	<i>Transformers</i> e Geração Autoregressiva	21
2.1.2	Modelos Servidos por Provedor	21
2.1.3	<i>Harness</i> como Camada de Engenharia	22
2.2	GERAÇÃO AUMENTADA POR RECUPERAÇÃO	22
2.2.1	RAG Documental e Busca Vetorial	22
2.2.2	RAG Estruturado e Dados Operacionais	23
2.3	AGENTES COM FERRAMENTAS	23
2.3.1	<i>Tool Calling</i>	24
2.3.2	<i>Model Context Protocol</i>	24
2.4	ORQUESTRAÇÃO EM AGENTES CORPORATIVOS	25
2.4.1	Roteamento e Resolução de Entidades	25
2.4.2	Execução, Observação e Composição	25
2.5	TECNOLOGIAS E REPRESENTAÇÃO DOS DADOS	26
2.6	CONTEXTO LONGO, <i>TOKENS</i> E DEGRADAÇÃO	26
2.7	ESTRATÉGIAS DE CONTROLE DE CONTEXTO	27
2.7.1	Seleção de Ferramentas	27
2.7.2	Compressão de <i>Prompts</i> e <i>Payloads</i>	27
2.7.3	Memória e Estado Compacto	27
2.7.4	Separação entre Evidência e Resolução	28
2.8	<i>BENCHMARK HARNESS</i> E ABLAÇÃO	28
2.9	SÍNTESE DO CAPÍTULO	29
3	DESCRIÇÃO DO PROBLEMA E REQUISITOS TÉCNICOS	30
3.1	CONTEXTO OPERACIONAL DA LIA	30
3.2	RECORTE EXPERIMENTAL E AMBIENTE LOCAL	31
3.3	FONTES, MCPS E FERRAMENTAS DISPONÍVEIS	32

3.4	CARACTERIZAÇÃO TÉCNICA DO PROBLEMA	33
3.5	LIMITAÇÕES DO RECORTE LOCAL	34
3.6	REQUISITOS INTEGRADOS DO SISTEMA AVALIADO	35
3.7	CAPACIDADES AVALIADAS PELO BENCHMARK	36
3.8	SÍNTESE DO CAPÍTULO	37
4	SOLUÇÃO PROPOSTA E METODOLOGIA UTILIZADA	38
4.1	VISÃO GERAL DA SOLUÇÃO	38
4.2	LINHA DE BASE V2 DA LIA	39
4.3	INTEGRAÇÃO COM MCPS E FONTES DE DADOS	39
4.4	ARQUITETURAS DE OTIMIZAÇÃO IMPLEMENTADAS	40
4.4.1	Restrição Semântica de Ferramentas	41
4.4.2	Compact Payload	42
4.4.3	AutoTool	43
4.4.4	TeaRAG	45
4.4.5	Membox	45
4.4.6	Agent Cognitive Compressor	46
4.4.7	<i>Retrieve-then-Solve</i>	48
4.5	CONFIGURAÇÃO DAS ABLAÇÕES	48
4.6	DESENHO DOS TESTES DE BENCHMARK	49
4.7	CRITÉRIOS DE AVALIAÇÃO	50
4.8	ATENDIMENTO AOS REQUISITOS	51
4.9	SÍNTESE DO CAPÍTULO	52
5	DESENVOLVIMENTO E ANÁLISE DOS RESULTADOS OBTIDOS .	53
5.1	RECORTE DA ANÁLISE E CRITÉRIOS DE COMPARAÇÃO	53
5.2	CUSTO TÉCNICO, PREÇO POR TOKEN E EFICIÊNCIA ECONÔMICA	55
5.3	PORTE, PREÇO POR TOKEN E CONTROLE DE VIÉS ENTRE MO- DELOS	57
5.4	LEITURA GERAL: GANHO ARQUITETURAL NÃO UNIVERSAL . .	61
5.5	RESULTADOS POR ARQUITETURA	62
5.6	MELHOR ARQUITETURA POR TESTE	63
5.7	ADERÊNCIA ENTRE HIPÓTESE ARQUITETURAL E RESULTADO PRÁTICO	65
5.8	ANÁLISE DETALHADA POR MODELO	66
5.8.1	Gemma4 31B: ganho simultâneo em acerto, <i>tokens</i> e latência . .	66
5.8.2	Nemotron 3 Super: máximo acerto com ACC, eficiência com MB+STR	67
5.8.3	GPT-OSS 20B: <i>harness</i> como compensação de capacidade . . .	68
5.8.4	MiniMax M2.7: teto alto, controle D1 e troca de perfil	70
5.8.5	GPT-OSS 120B: <i>baseline</i> forte e <i>harness</i> seletivo	71

5.8.6	DeepSeek V4 Flash: maior acerto com regressão de custo e segurança	71
5.8.7	Kimi K2.6: ganho pontual em L6 e L9 com AT+MB	72
5.8.8	MiniMax M3: eficiência na rodada completa e ganho textual na parcial	73
6	CONCLUSÃO	77
6.1	SÍNTESE DO TRABALHO	77
6.2	ATENDIMENTO AOS OBJETIVOS	77
6.3	PRINCIPAIS RESULTADOS	78
6.4	LIMITAÇÕES E PRÓXIMOS PASSOS	79
6.5	IMPACTOS DO TRABALHO	80
	REFERÊNCIAS	82
	APÊNDICE A – DESCRIÇÃO 1	85
	ANEXO A – DESCRIÇÃO 2	86

1 INTRODUÇÃO

1.1 CONTEXTUALIZAÇÃO

A incorporação de soluções de Inteligência Artificial (IA) generativa em ambientes corporativos tem alterado a forma como informações internas e externas são consultadas para apoio à decisão. No setor de saúde, essa mudança é especialmente relevante porque análises comerciais dependem da combinação de bases heterogêneas: cadastros de estabelecimentos, informações assistenciais, dados empresariais, registros de relacionamento com clientes, equipamentos médicos e características territoriais.

O contexto que motivou este trabalho está associado à Lifeshub e à aplicação LIA, uma plataforma de inteligência conversacional orientada a dados. A aplicação foi desenvolvida para apoiar atividades de inteligência comercial e suporte a vendas complexas no setor de saúde. Nesse tipo de operação, uma equipe comercial pode precisar responder rapidamente a perguntas como: qual é o perfil de determinado hospital, qual entidade jurídica o mantém, se há registro de relacionamento no *Customer Relationship Management* (CRM), quais dados assistenciais estão disponíveis e quais lacunas impedem uma conclusão segura.

Tradicionalmente, esse processo exige navegação por planilhas, relatórios tabulares, *dashboards* e sistemas distintos. Embora esses instrumentos sejam úteis para análises recorrentes, eles dependem de filtros previamente conhecidos e exigem que o analista conecte manualmente entidades entre bases diferentes. Uma pergunta aparentemente simples sobre um hospital pode envolver Cadastro Nacional de Estabelecimentos de Saúde (CNES), dados empresariais, Quadro de Sócios e Administradores (QSA), CRM e informações territoriais.

A LIA atua como uma camada de orquestração sobre essas fontes. Em vez de substituir as bases existentes, o agente recebe uma pergunta em linguagem natural, seleciona ferramentas externas, recupera evidências e compõe uma resposta. O valor técnico da aplicação, portanto, não está apenas na geração de texto, mas na capacidade de decidir quais fontes consultar, em que ordem, com quais parâmetros e quando declarar que determinada informação não está disponível.

O recorte deste trabalho considera modelos de linguagem consumidos por provedores ou interfaces compatíveis com serviços de nuvem. Assim, não são abordadas técnicas internas ao modelo, como *fine-tuning*, quantização, treinamento, destilação ou otimização do mecanismo de predição do próximo *token*. O foco está no *harness* ao redor do modelo: organização do *prompt*, seleção de ferramentas, controle de contexto, memória, compactação de resultados, decomposição de tarefas e avaliação experimental.

1.2 DEFINIÇÃO DO PROBLEMA

Agentes baseados em *Large Language Models* (LLMs) com *tool calling* enfrentam um problema de engenharia quando precisam operar com muitas ferramentas e múltiplos turnos. Cada chamada ao modelo pode carregar instruções, histórico, descrições de ferramentas, observações anteriores e resultados intermediários. Esse contexto aumenta custo, latência e risco de perda de foco.

No caso da LIA, o problema é agravado pela necessidade de cruzar domínios. O agente precisa preservar a entidade correta ao transitar entre estabelecimento de saúde, CNPJ, empresa mantenedora, governança societária, conta de CRM e dados territoriais. Uma resposta pode parecer fluente e ainda estar errada se consultar a entidade incorreta, omitir uma ferramenta obrigatória ou inferir informação que a base local não sustenta.

Este trabalho não propõe o desenvolvimento da plataforma LIA em si, pois ela já se encontra funcional. O recorte adotado consiste em implementar e avaliar novas arquiteturas de orquestração sobre essa base, envolvendo seleção de ferramentas, controle de contexto, organização de evidências, memória e critérios de avaliação experimental.

O problema central deste trabalho pode ser formulado da seguinte forma: como reduzir consumo de *tokens*, latência e instabilidade em um agente de IA generativa com ferramentas externas, preservando qualidade factual em um cenário de integração de dados de saúde e suporte a vendas complexas?

Essa formulação considera também as limitações do ambiente experimental. A versão local da LIA não contém a totalidade das bases e campos disponíveis no sistema completo. Por isso, os testes precisam medir não apenas se o agente encontra respostas, mas também se reconhece lacunas de cobertura e evita produzir conclusões sem evidência.

1.3 OBJETIVOS

1.3.1 Objetivo Geral

O objetivo geral deste trabalho é desenvolver e avaliar otimizações arquiteturais para a plataforma LIA, buscando reduzir consumo de *tokens*, latência e instabilidade operacional em agentes de IA generativa com ferramentas, sem comprometer a qualidade factual das respostas em cenários de suporte a vendas complexas.

1.3.2 Objetivos Específicos

Para atingir esse objetivo, foram definidos os seguintes objetivos específicos:

- analisar a arquitetura existente da LIA e identificar pontos de custo, latência e degradação factual;
- estruturar um ambiente experimental local e controlado, representativo do problema de orquestração, sem depender do sistema completo de produção;
- desenvolver uma suíte de *benchmark* com consultas diretas, cadeias multi-etapas, cruzamento de domínios, retenção de contexto e controles de honestidade;
- implementar variantes arquiteturais para seleção de ferramentas, compactação de *payload*, inferência de sequências, memória, compressão cognitiva e separação entre recuperação e resolução;
- comparar arquiteturas e modelos por métricas de acerto, nota factual, latência, *tokens* de entrada, *tokens* de saída e ferramentas acionadas;
- identificar em quais condições cada arquitetura melhora ou prejudica o comportamento do agente.

1.4 ESCOPO E DELIMITAÇÕES

Este trabalho concentra-se na camada de orquestração da LIA e no *benchmark harness* usado para avaliá-la. A preparação das bases por processos de *Extract, Transform and Load* (ETL), a interface de usuário, módulos administrativos, autenticação e funcionalidades corporativas auxiliares não são avaliados em profundidade.

Também ficam fora do escopo otimizações internas dos modelos. Como os modelos avaliados são consumidos como serviço, a aplicação não controla pesos, treinamento, quantização, infraestrutura de inferência ou algoritmo de geração. As otimizações estudadas são externas ao modelo e podem ser aplicadas independentemente do provedor, desde que a interface permita mensagens, ferramentas e coleta de métricas.

O ambiente local utilizado é parcial. Algumas bases do sistema completo não foram materializadas localmente por tamanho, custo ou dependência de infraestrutura. Essa limitação restringe a variedade dos testes e torna necessário distinguir perguntas respondíveis de perguntas que devem ser recusadas ou qualificadas por falta de cobertura.

Por fim, a execução de ablações com múltiplos modelos e arquiteturas é limitada por cotas de provedor, tempo de execução e instabilidade de serviços. O *benchmark* foi adaptado para salvar *checkpoints* e separar falhas de infraestrutura de falhas funcionais do agente.

1.5 SOLUÇÃO PROPOSTA E METODOLOGIA GERAL

A solução parte da arquitetura V2 da LIA, composta por roteamento, resolução de entidades, execução de ferramentas, composição de resposta e controle de estado. Sobre essa base foi construído um *benchmark harness* capaz de executar a mesma suíte de perguntas com diferentes modelos e variantes arquiteturais.

As variantes investigadas atuam em pontos distintos do fluxo: seleção semântica de ferramentas, compactação de resultados, antecipação de sequências de ferramentas, memória por tópicos, compressão cognitiva do estado e separação entre etapa de extração de evidência e etapa de resolução. A avaliação foi conduzida por ablação, mantendo perguntas, dados e critérios constantes enquanto combinações arquiteturais eram ativadas ou removidas.

As execuções registram latência, ferramentas acionadas, *tokens* de entrada, *tokens* de saída, resposta final, status de qualidade e falhas de infraestrutura. Com isso, a comparação deixa de depender apenas da aparência da resposta e passa a considerar também custo, rastreabilidade e comportamento do agente ao longo da execução.

1.6 CONTRIBUIÇÕES DO TRABALHO

As principais contribuições do trabalho são:

- caracterização de um problema real de orquestração de agentes com ferramentas em bases heterogêneas de saúde e negócio;
- construção de uma suíte de *benchmark* discriminativa para avaliar recuperação factual, encadeamento, memória, uso de ferramentas e honestidade sobre dados ausentes;
- implementação controlada de variantes arquiteturais sobre uma linha de base funcional da LIA;
- comparação quantitativa entre modelos e arquiteturas por acerto, custo, latência e rastreabilidade;
- identificação de limitações práticas associadas a cobertura de dados, custo de *tokens*, *timeouts* e métricas incompletas.

1.7 ESTRUTURA DO DOCUMENTO

Este documento está organizado em seis capítulos. O Capítulo 2 apresenta a fundamentação teórica sobre LLMs, *Retrieval-Augmented Generation* (RAG), agentes

com ferramentas, protocolos de integração, gestão de contexto e avaliação experimental. O Capítulo 3 descreve o problema técnico, o recorte da LIA, as fontes disponíveis e os requisitos do sistema avaliado. O Capítulo 4 detalha a solução proposta e a metodologia utilizada, incluindo o fluxo de orquestração, as arquiteturas implementadas, a configuração das ablações, os testes e os critérios de avaliação. O Capítulo 5 apresenta o desenvolvimento concreto e a análise dos resultados obtidos. Por fim, o Capítulo 6 sintetiza as conclusões, limitações e possibilidades de trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta os conceitos técnicos necessários para compreender o desenvolvimento realizado. A fundamentação não busca descrever todos os temas de IA generativa, mas sim os elementos diretamente relacionados ao problema deste trabalho: modelos de linguagem servidos por provedor, agentes com ferramentas, recuperação de evidências, orquestração, controle de contexto e avaliação por ablação.

Alguns termos são mantidos em inglês por serem os nomes predominantes na literatura e nas ferramentas usadas na implementação, como *prompt*, *tool calling*, *router*, *payload*, *embedding*, *vector database*, *benchmark* e *harness*. Quando necessário, o texto apresenta o significado operacional desses termos no contexto da LIA.

2.1 MODELOS DE LINGUAGEM DE LARGA ESCALA

2.1.1 *Transformers* e Geração Autoregressiva

Modelos de linguagem de larga escala, ou LLMs, são modelos generativos treinados para estimar distribuições de probabilidade sobre sequências de *tokens*. A arquitetura *Transformer*, proposta por Vaswani et al. (2017), tornou-se uma das bases técnicas desses modelos ao substituir mecanismos recorrentes por mecanismos de atenção, permitindo que o modelo relacione diferentes posições da sequência durante o processamento.

Em modelos autoregressivos modernos, instruções, exemplos, mensagens do usuário, resultados de ferramentas e respostas parciais são representados como *tokens* em uma janela de contexto. Brown et al. (2020) demonstram que modelos de grande escala podem executar tarefas a partir de instruções e poucos exemplos no próprio *prompt*, característica conhecida como aprendizado em contexto. Essa capacidade é central para aplicações corporativas, pois permite adaptar o comportamento do modelo em tempo de inferência, sem retreinar seus pesos.

Apesar dessa flexibilidade, o conhecimento do modelo é principalmente paramétrico. Em aplicações empresariais, esse conhecimento raramente é suficiente: a resposta correta depende de bases atualizadas, identificadores internos, regras de negócio, registros privados e informações que podem não estar presentes no treinamento. No domínio deste trabalho, perguntas sobre estabelecimentos de saúde, empresas mantenedoras, QSA, equipamentos e CRM exigem acesso explícito a fontes externas.

2.1.2 Modelos Servidos por Provedor

As possibilidades de otimização dependem do nível de controle sobre o modelo. Quando a organização executa modelos localmente, podem ser consideradas técnicas como *fine-tuning*, quantização, escolha de formato de pesos, otimização de *serving*,

ajuste de *batching* ou mudanças na infraestrutura de inferência. Essas técnicas atuam no próprio modelo ou na forma como ele é servido.

O escopo deste trabalho é diferente. Os modelos avaliados são consumidos por APIs ou interfaces compatíveis com provedores de nuvem. Portanto, não há controle direto sobre pesos, treinamento, quantização, arquitetura interna do *Transformer* ou mecanismo de predição do próximo *token*. A otimização possível está no *harness*: instruções, estrutura do *prompt*, seleção de ferramentas, compactação de *payload*, memória, decomposição de etapas e critérios de avaliação.

Essa delimitação é importante porque representa um cenário comum em aplicações corporativas. A empresa pode trocar de modelo, ajustar *prompts* e controlar ferramentas, mas nem sempre pode treinar ou hospedar um modelo próprio. O trabalho, portanto, investiga como extrair melhor desempenho de modelos já disponíveis como serviço, reduzindo desperdício de *tokens* e chamadas sem abandonar a rastreabilidade.

2.1.3 *Harness* como Camada de Engenharia

O termo *harness* é usado neste trabalho para designar o conjunto de mecanismos que envolve o modelo durante sua execução. Ele inclui o *template* de mensagens, as instruções de sistema, o catálogo de ferramentas, o formato dos retornos, a memória da conversa, os critérios de parada, a coleta de métricas e o código que executa os testes.

Em um agente com ferramentas, o desempenho observado não depende apenas da capacidade do modelo. Um modelo forte pode compensar um *prompt* longo ou um catálogo de ferramentas mal organizado, mas a custo maior. Um modelo menor pode falhar em um fluxo pouco estruturado e melhorar quando recebe ferramentas corretas, evidências compactas e instruções claras. Por isso, o *harness* é parte da solução, não apenas uma camada auxiliar.

2.2 GERAÇÃO AUMENTADA POR RECUPERAÇÃO

2.2.1 RAG Documental e Busca Vetorial

A geração aumentada por recuperação, conhecida como RAG, combina modelos generativos com mecanismos de busca em bases externas. Em vez de depender apenas do conhecimento paramétrico do modelo, um sistema RAG recupera documentos ou registros relevantes e os insere no contexto usado para gerar a resposta. Lewis et al. (2020) formalizam essa abordagem ao combinar memória paramétrica, presente no modelo, com memória não paramétrica, representada por um índice consultável de documentos.

Em fluxos documentais, textos são divididos em trechos e transformados em vetores numéricos chamados *embeddings*. Esses vetores são armazenados em uma *vector database* ou mecanismo equivalente. Quando o usuário faz uma pergunta, a pergunta também é vetorizada, e o sistema recupera trechos semanticamente próximos. Esse processo não equivale a ler uma base inteira: ele é uma busca aproximada, sujeita a problemas de segmentação, ranqueamento, cobertura e ambiguidade.

Essa limitação é relevante para sistemas corporativos. Um trecho recuperado pode ser parecido com a pergunta, mas não conter a resposta; pode conter a resposta de entidade semelhante; ou pode omitir um campo necessário. Assim, a qualidade do RAG depende tanto da recuperação quanto da capacidade do agente de verificar se a evidência recuperada realmente sustenta a conclusão.

2.2.2 RAG Estruturado e Dados Operacionais

No contexto da LIA, o conceito de RAG é ampliado para além da recuperação de textos. A aplicação precisa consultar registros estruturados e ferramentas de domínio: perfil de estabelecimento, empresa mantenedora, governança, conta de CRM, equipamentos e dados territoriais. Nesse caso, a evidência não é apenas um parágrafo; pode ser um objeto *JavaScript Object Notation* (JSON) retornado por uma ferramenta.

Por exemplo, se a pergunta solicita o perfil do estabelecimento com CNES 2077485, o agente deve consultar uma ferramenta de cadastro estruturado, recuperar nome, CNPJ, natureza jurídica, município e leitos, e somente então compor a resposta. Em perguntas mais complexas, esse resultado se torna entrada para outra etapa, como consultar a empresa associada ao CNPJ ou verificar o QSA.

Essa forma de recuperação aproxima o sistema de um RAG *agentic*, no qual um agente controla a recuperação de informações e decide dinamicamente como buscar, reconsultar, cruzar e sintetizar evidências. O TeaRAG, proposto por Chao Zhang et al. (2025), parte do problema de custo em agentes RAG e busca reduzir conteúdo recuperado por meio de relações em grafo. A contribuição relevante para este trabalho é a ideia de que evidências intermediárias devem ser controladas e compactadas, não simplesmente acumuladas.

2.3 AGENTES COM FERRAMENTAS

Um agente baseado em LLM pode ser entendido como um sistema que usa o modelo para decidir ações, observar seus resultados e produzir uma resposta final. Diferentemente de um *chatbot* simples, o agente pode executar chamadas intermediárias para obter dados, validar hipóteses ou transformar informações. Essa capacidade é essencial quando a tarefa envolve bases privadas, identificadores estruturados ou cadeias de consulta.

A literatura recente descreve esse tipo de sistema como modelo de linguagem aumentado por ferramentas. Mialon et al. (2023) organizam essa família de abordagens em torno da ampliação de modelos com recuperação, raciocínio e uso de ferramentas. O paradigma ReAct, proposto por Yao et al. (2023), intercala raciocínio textual e ações, permitindo que o modelo atualize seu plano a partir das observações retornadas pelo ambiente. Já Schick et al. (2023) investigam como modelos podem aprender quando chamar APIs, quais argumentos usar e como incorporar os resultados.

2.3.1 *Tool Calling*

No *tool calling*, cada ferramenta é apresentada ao modelo com nome, descrição, contrato de entrada e, em geral, um *schema* em JSON. O modelo não executa a ferramenta diretamente; ele gera uma chamada estruturada. A aplicação recebe essa chamada, executa a função real no *backend* ou em um servidor externo e devolve a observação ao modelo.

Esse mecanismo cria um canal estruturado dentro da conversa. Em vez de responder apenas com texto livre, o modelo pode solicitar, por exemplo, a execução de `perfil_estabelecimento` com o argumento `cnes = 2077485`. A aplicação executa a consulta, retorna um objeto estruturado, e o modelo usa esse objeto para compor a resposta. Essa separação é fundamental para auditoria, pois permite verificar se uma afirmação foi sustentada por ferramenta.

O desafio não está apenas em chamar uma ferramenta, mas em escolher a ferramenta correta e preencher seus parâmetros. Consultar diretamente um perfil por CNES é diferente de buscar uma unidade por nome aproximado; consultar governança empresarial é diferente de consultar dados assistenciais; verificar conta no CRM é diferente de inferir relacionamento comercial a partir de dados públicos.

2.3.2 *Model Context Protocol*

À medida que sistemas *agentic* passam a depender de muitas ferramentas, torna-se necessário separar a lógica do agente da implementação de cada fonte de dados. Um protocolo de ferramentas define como clientes e servidores anunciam capacidades, descrevem funções, recebem chamadas e devolvem resultados.

O *Model Context Protocol* (MCP) é um protocolo aberto para padronizar a comunicação entre aplicações de IA e servidores que expõem contexto, recursos, *prompts* e ferramentas. A especificação define uma arquitetura cliente-servidor baseada em mensagens JSON-RPC, negociação de capacidades e ciclo de vida de conexão (Model Context Protocol, 2025). No escopo deste trabalho, essa padronização permite representar bases de saúde, empresas, equipamentos e CRM como servidores especializados.

Um exemplo concreto é a ferramenta `perfil_estabelecimento`. Para o modelo, ela aparece como uma capacidade com nome, descrição e argumento obrigatório. Para a aplicação, essa chamada encapsula validação de parâmetro, consulta à fonte correta, normalização do resultado e retorno em JSON. Essa fronteira permite que o agente use a informação sem conhecer a consulta física.

2.4 ORQUESTRAÇÃO EM AGENTES CORPORATIVOS

A orquestração é a camada que transforma uma pergunta em uma sequência controlada de decisões. Em agentes corporativos, essa camada costuma incluir roteamento de intenção, resolução de entidades, seleção de ferramentas, execução de chamadas, tratamento de erros, memória e composição final. No caso da LIA, a orquestração é o ponto central do trabalho, pois é nela que se decide quanto contexto expor ao modelo e quais ferramentas permitir em cada etapa.

2.4.1 Roteamento e Resolução de Entidades

O *router* identifica o tipo de tarefa solicitada. Uma pergunta simples por CNES pode seguir um caminho direto de consulta cadastral. Um dossiê exige múltiplos domínios. Uma pergunta territorial pode exigir verificação de cobertura antes de tentar responder. O roteamento reduz chamadas desnecessárias e evita que o modelo receba ferramentas irrelevantes.

A resolução de entidades complementa o roteamento. Ela transforma nomes, identificadores e referências de conversa em entidades canônicas. Esse passo é essencial porque a mesma pergunta pode mencionar “Hospital São Paulo”, “Hospital de Ensino da Unifesp” ou diretamente CNES 2077485. Sem normalização, o agente pode trocar a instituição por outra entidade conhecida ou usar CNPJ incompleto.

2.4.2 Execução, Observação e Composição

Após escolher ferramentas, o agente precisa interpretar seus retornos. Ferramentas podem retornar objetos completos, listas de candidatos, registros vazios ou erros. A composição final deve transformar essas observações em resposta clara, preservando evidências e declarando lacunas.

Essa etapa é particularmente sensível em perguntas comerciais. Se a ferramenta de CRM não retorna conta correspondente, o agente não deve concluir que a empresa nunca teve relacionamento; deve declarar que não foi encontrado registro no recorte consultado. A composição final precisa ser útil, mas também epistemicamente honesta.

2.5 TECNOLOGIAS E REPRESENTAÇÃO DOS DADOS

A implementação da LIA utiliza *backend* em Node.js, NestJS e TypeScript, com orquestração baseada em LangGraph.js. O *backend* controla sessões, executa o grafo do agente, comunica-se com modelos de linguagem e aciona servidores MCP. A infraestrutura local é organizada com Docker Compose, permitindo subir serviços de aplicação, bancos, índices e servidores auxiliares de forma controlada.

As bases utilizadas combinam diferentes formas de armazenamento. O PostgreSQL apoia dados operacionais, como sessões, usuários, mensagens e *checkpoints*. O Elasticsearch é usado para busca textual, busca vetorial e índices operacionais de baixa latência. O DuckDB sobre arquivos Parquet é adequado para leitura analítica de grandes volumes materializados, especialmente quando a pergunta exige agregações ou consultas sobre dados de *lakehouse*. O JSON aparece como formato de troca entre ferramentas, *backend* e modelo.

Essas tecnologias têm papéis distintos dentro do agente. Uma busca por nome de hospital tende a ser mais adequada ao Elasticsearch. Uma consulta de perfil por identificador tende a ser mais segura por ferramenta tipada. Uma agregação sobre grande volume pode exigir DuckDB/Parquet, desde que a consulta seja restrita e orientada pelo dicionário de dados. O *frontend* não é aprofundado neste trabalho, pois a contribuição avaliada está no *backend* de orquestração e no *benchmark harness*.

2.6 CONTEXTO LONGO, TOKENS E DEGRADAÇÃO

A janela de contexto é um recurso finito. Em um agente com ferramentas, ela pode ser consumida por instruções do sistema, mensagem do usuário, histórico, descrições de ferramentas, resultados de chamadas, memórias intermediárias e resposta final. Mesmo quando o modelo aceita uma grande quantidade de *tokens*, contextos extensos aumentam custo e latência.

O problema não é apenas quantitativo. Nelson F. Liu et al. (2024) mostram que modelos podem ter dificuldade em utilizar informações presentes no meio de contextos longos. Esse resultado indica que anexar mais documentos, mais histórico ou mais observações não garante melhor resposta. Pelo contrário, informação irrelevante pode diluir restrições e reduzir a capacidade do modelo de focar na evidência correta.

Em conversas multi-turno, a estratégia ingênua consiste em reenviar todo o *transcript* a cada chamada. Embora preserve informação, esse método acumula ruído: tentativas abandonadas, erros anteriores, resultados redundantes e instruções antigas. Em tarefas longas, esse acúmulo pode provocar desvio do objetivo, esquecimento de restrições e uso de observações que já não são relevantes.

2.7 ESTRATÉGIAS DE CONTROLE DE CONTEXTO

2.7.1 Seleção de Ferramentas

Uma forma de controlar contexto é selecionar apenas as ferramentas relevantes para cada tarefa. Em vez de enviar ao modelo todo o catálogo, o sistema pode representar ferramentas em um espaço semântico e recuperar as mais próximas da intenção atual. Essa estratégia reduz *tokens* e pode diminuir confusão, mas introduz um novo risco: se a ferramenta necessária não for recuperada, o agente fica impedido de resolver a tarefa.

A literatura de recuperação de ferramentas discute esse compromisso. O ToolRerank propõe reranqueamento adaptativo e sensível à hierarquia da biblioteca de ferramentas (Zheng et al., 2024). O COLT argumenta que sistemas reais precisam recuperar conjuntos completos de ferramentas complementares, não apenas a ferramenta semanticamente mais próxima (Qu et al., 2024). Essa preocupação é central para a LIA, pois uma pergunta pode exigir saúde, empresa e CRM na mesma cadeia.

2.7.2 Compressão de *Prompts* e *Payloads*

Outra família de técnicas reduz diretamente o volume de texto ou dados enviados ao modelo. O LLMLingua propõe compressão de *prompts* por controle de orçamento e compressão em nível de *token* (Jiang et al., 2023). O LongLLMLingua adapta a ideia a cenários de contexto longo (Jiang et al., 2024). O HyCo2 discute compressão híbrida para preservar informações locais e globais (Liao et al., 2025).

Na LIA, a compactação estudada é uma adaptação de engenharia para *payloads* estruturados. Em vez de aplicar um compressor neural genérico, a implementação remove campos vazios, valores nulos, listas redundantes e excesso de resultados antes que o retorno da ferramenta seja reinjetado no *prompt*. A vantagem é previsibilidade e auditoria; a limitação é que regras determinísticas podem remover campos úteis se forem agressivas demais.

2.7.3 Memória e Estado Compacto

Além de selecionar ferramentas e compactar *payloads*, agentes podem substituir o histórico bruto por representações de estado. Uma memória compacta pode preservar objetivo atual, entidades relevantes, restrições, evidências confirmadas e próximos passos. Isso reduz o *replay* de *transcript* e pode melhorar continuidade em conversas longas.

Entretanto, memória também introduz risco. Se uma restrição importante for perdida ou se uma entidade for consolidada incorretamente, o erro passa a condicionar turnos futuros. Levantamentos recentes distinguem memória de simples recuperação,

pois memória envolve atualização, persistência, esquecimento seletivo e controle do que passa a influenciar decisões futuras (Hu, Yuyang et al., 2026; Hu, Yuanzhe; Wang; McAuley, 2026).

O Membox propõe continuidade por tópicos em diálogos longos (Tao et al., 2026). O MEM1 propõe memória constante para tarefas longas (Zhou et al., 2025). Neste trabalho, o MEM1 foi mantido como referência conceitual, mas não como arquitetura principal de análise, pois sua implementação local não foi estabilizada no mesmo nível das demais variantes. Já o *Agent Cognitive Compressor* (ACC) fundamenta a ideia de substituir *replay* de *transcript* por um estado cognitivo comprimido e estruturado (Bousetouane, 2026).

2.7.4 Separação entre Evidência e Resolução

Outra estratégia é separar a recuperação da solução. O estudo de Du et al. (2025) mostra que o desempenho pode cair com o aumento do contexto mesmo quando a evidência correta está presente. A técnica *Retrieve-then-Solve* divide a tarefa em duas chamadas: primeiro extrair ou recitar evidências relevantes; depois resolver a pergunta em um contexto curto.

Essa técnica é compatível com modelos servidos por provedor, pois atua apenas no nível do *prompt*. No entanto, ela adiciona uma chamada ao modelo e depende da qualidade da evidência extraída. Se a primeira etapa omite um fato necessário, a segunda etapa fica limitada. No *benchmark*, essa abordagem é tratada como encurtamento cognitivo, não como garantia de redução de custo total.

2.8 BENCHMARK HARNESS E ABLAÇÃO

A avaliação de agentes com ferramentas difere da avaliação de respostas isoladas. Uma resposta pode parecer correta, mas ter sido obtida com entidade errada, ferramenta inadequada ou evidência insuficiente. Por isso, um *benchmark* útil precisa observar resultado e percurso.

O *benchmark harness* usado neste trabalho executa cenários padronizados, registra ferramentas acionadas, mede *tokens*, salva *traces* e classifica a qualidade da resposta por critérios explícitos. Ele permite comparar modelos e arquiteturas sem depender apenas da percepção qualitativa do usuário.

A ablação é o método usado para comparar variantes. Uma configuração base é preservada, e mecanismos são ativados ou removidos de forma controlada. No caso da LIA, isso permite comparar a V2 com variantes que incluem seleção semântica de ferramentas, compactação de *payload*, AutoTool, Membox, ACC e *Retrieve-then-Solve*. O objetivo não é encontrar uma arquitetura universalmente superior, mas entender em quais condições cada mecanismo melhora ou prejudica acerto, custo, latência e

estabilidade.

O Quadro 1 resume os conceitos apresentados e sua função no restante do trabalho.

Quadro 1 – Conceitos teóricos utilizados no trabalho

Conceito	Papel no trabalho
LLM	Motor generativo responsável por interpretar perguntas, decidir ações e produzir linguagem natural.
RAG	Estratégia para fundamentar respostas em evidências externas ao conhecimento paramétrico do modelo.
<i>Tool calling</i>	Mecanismo que permite ao modelo solicitar chamadas estruturadas a ferramentas externas.
MCP	Protocolo usado como referência para modularizar servidores de ferramentas e fontes de dados.
Orquestração	Controle da sequência entre roteamento, resolução de entidades, ferramentas, observações e resposta final.
<i>Stack</i> de dados	Combinação de PostgreSQL, Elasticsearch, DuckDB, Parquet e JSON para suportar memória, busca e consultas estruturadas.
Gestão de contexto	Controle de histórico, ferramentas, evidências e memória para reduzir <i>tokens</i> e evitar perda de foco.
Seleção e compressão	Estratégias para reduzir ferramentas ou <i>payloads</i> expostos ao modelo preservando evidências críticas.
Memória compacta	Representação de estado usada para preservar continuidade sem reenviar todo o histórico.
<i>Benchmark harness</i>	Bancada de execução padronizada para medir acerto, custo, latência, ferramentas e estabilidade.

Fonte: Elaboração própria.

2.9 SÍNTESE DO CAPÍTULO

Este capítulo apresentou os fundamentos necessários para compreender a arquitetura estudada: modelos de linguagem, RAG, agentes com ferramentas, MCP, orquestração, tecnologias de dados, custo de contexto, estratégias de controle e avaliação por ablação. A partir desses conceitos, o próximo capítulo descreve o problema específico da LIA, o ambiente experimental disponível e os requisitos que orientaram a solução.

3 DESCRIÇÃO DO PROBLEMA E REQUISITOS TÉCNICOS

Este capítulo descreve o problema técnico tratado no projeto, o recorte da plataforma LIA utilizado nos experimentos e os requisitos que orientaram a solução. A intenção é caracterizar o ambiente antes de apresentar as arquiteturas propostas: quais fontes estavam disponíveis, quais ferramentas podiam ser acionadas, quais limitações existiam no recorte local e que tipo de comportamento o *benchmark* precisava medir.

Essa separação é importante porque o desempenho de um agente com ferramentas não depende apenas do modelo de linguagem. O resultado final também é afetado pelo catálogo de ferramentas, pela qualidade das descrições, pela cobertura das bases, pela forma como o contexto é montado, pelo limite de *tokens*, pela latência dos provedores e pelo modo como falhas são registradas. Portanto, o problema deste trabalho não é simplesmente perguntar se um modelo “sabe” responder, mas avaliar se diferentes arquiteturas de orquestração ajudam o modelo a usar melhor os recursos disponíveis.

3.1 CONTEXTO OPERACIONAL DA LIA

A LIA foi concebida como uma camada conversacional para apoiar consultas sobre dados de saúde, inteligência comercial e relacionamento com instituições. No cenário estudado, o usuário não procura apenas recuperar um campo isolado de uma tabela. Em geral, a necessidade é transformar dados distribuídos em uma resposta útil para análise de uma conta, preparação comercial, qualificação de oportunidade ou investigação cadastral de uma instituição.

Uma pergunta aparentemente simples sobre um hospital pode envolver diferentes entidades e bases. O estabelecimento é identificado no domínio de saúde por um código CNES; a entidade responsável pode exigir a extração de um CNPJ; a caracterização jurídica depende de dados empresariais; a governança pode exigir consulta ao QSA; a existência de relacionamento comercial precisa ser verificada em CRM; e dados territoriais ou demográficos dependem de cobertura municipal ou regional. O problema, portanto, não é somente localizar um documento correto, mas preservar a entidade correta enquanto o agente atravessa essas camadas.

Esse cenário transforma a LIA em um problema de orquestração. O agente precisa interpretar a pergunta, decidir se há necessidade de ferramentas externas, escolher a ferramenta adequada, preencher parâmetros, usar o retorno para a próxima etapa e compor uma resposta que diferencie fatos recuperados de lacunas de cobertura. Se uma dessas etapas falha, a resposta pode continuar parecendo plausível, mas deixar de ser confiável para uso operacional.

O exemplo recorrente deste trabalho é o Hospital São Paulo, identificado pelo

CNES 2077485. Esse caso foi usado porque permite exercitar diferentes níveis de dificuldade: consulta direta do perfil do estabelecimento, resolução textual por nome, cruzamento com base empresarial, busca de sócios, consulta de CRM, análise de equipamentos, manutenção de contexto em múltiplos turnos e reconhecimento de limitações quando a base local não cobre determinada pergunta.

3.2 RECORTE EXPERIMENTAL E AMBIENTE LOCAL

Os experimentos foram executados em uma versão local e parcial da plataforma. Esse recorte foi escolhido por permitir iteração rápida, controle de execução e repetibilidade durante as campanhas de ablação. Ele preserva o núcleo do problema de orquestração, mas não representa a totalidade do ambiente corporativo completo, que possui mais fontes, maior volume de dados, credenciais de produção e integrações não utilizadas neste trabalho.

No recorte utilizado, o *backend* da aplicação foi executado em Node.js, NestJS e TypeScript. A orquestração do agente utiliza LangGraph.js, com etapas de roteamento, resolução de entidades, execução de ferramentas, composição da resposta e salvamento de estado. A infraestrutura local usa Docker Compose para organizar serviços como PostgreSQL, Elasticsearch, serviços de *embedding*, *backend* da aplicação e servidores MCP.

As fontes consultadas pelas ferramentas combinam diferentes formas de armazenamento. O Elasticsearch é usado para buscas textuais e recuperação de registros de estabelecimentos. O DuckDB permite consultas analíticas sobre arquivos Parquet. O PostgreSQL aparece como base relacional de apoio. Arquivos JSON e CSV são usados em etapas de carga, configuração e avaliação. Essa diversidade reproduz uma característica comum de sistemas corporativos: a informação necessária não está em um único banco, nem em um formato uniforme.

O *pipeline* de ETL que materializa essas bases é considerado infraestrutura anterior ao experimento. Ele é relevante para explicar a origem dos dados, mas não é o objeto avaliado. Assim, este trabalho não mede a qualidade da extração, transformação ou carga dos dados; mede o comportamento do agente ao utilizar as ferramentas e bases já disponíveis no ambiente local.

Também é importante destacar que os modelos avaliados foram servidos por provedores de nuvem, mesmo quando eram modelos de pesos abertos. Isso limita o escopo das otimizações possíveis. Técnicas como *fine-tuning*, quantização, alteração da arquitetura interna do modelo ou otimização do processo de predição do próximo *token* não foram tratadas. O foco está no que pode ser alterado externamente: *prompts*, seleção de ferramentas, compactação de contexto, uso de memória, separação de etapas, critérios de parada e desenho do *benchmark harness*.

3.3 FONTES, MCPS E FERRAMENTAS DISPONÍVEIS

O agente avaliado não acessa tabelas internas livremente. Ele opera por capacidades expostas como ferramentas de domínio, em geral por servidores MCP. Essa decisão torna o ambiente mais controlável: cada fonte aparece para o agente como uma ação com nome, descrição, argumentos e retorno esperado. Em vez de enxergar uma estrutura completa de banco de dados, o modelo recebe uma interface de alto nível, por exemplo, consultar o perfil de um estabelecimento, buscar uma empresa por CNPJ ou verificar uma conta no CRM.

Essa mediação também reduz riscos. Quando o agente chama uma ferramenta tipada, o sistema consegue registrar a chamada, validar parâmetros, limitar escopo e padronizar o retorno. Por outro lado, ela cria um novo problema: se o modelo não escolher a ferramenta correta, ou se a descrição da ferramenta não for clara, a resposta pode falhar mesmo com os dados disponíveis. Por isso, o catálogo de ferramentas faz parte do objeto de avaliação.

O Quadro 2 resume os grupos de ferramentas relevantes para o escopo experimental. A tabela não lista todas as funções existentes no sistema, mas as capacidades que tiveram impacto direto no desenho dos testes e na interpretação dos resultados.

As ferramentas de estabelecimentos de saúde formam o núcleo dos testes porque permitem localizar a unidade, recuperar o perfil cadastral e obter atributos estruturais. Elas são suficientes para perguntas de busca direta, mas não bastam quando a pergunta exige governança, relacionamento comercial ou interpretação estratégica.

As ferramentas empresariais complementam a camada de saúde. Elas permitem transformar um CNPJ encontrado no perfil do estabelecimento em uma entidade jurídica analisável, com razão social, natureza jurídica e quadro societário. Esse salto entre domínios é um dos pontos centrais do *benchmark*, pois mede se o agente consegue usar um resultado intermediário como entrada correta para a etapa seguinte.

As ferramentas de CRM têm outro papel: verificar evidência comercial interna. Um erro importante seria tratar a existência de um hospital ou de uma empresa como prova de relacionamento comercial. O comportamento esperado é consultar a ferramenta adequada e, caso não exista retorno, declarar que não há evidência no recorte avaliado.

As ferramentas de municípios, demografia, equipamentos, profissionais e SQL ampliam o espaço de perguntas possíveis, mas também aumentam o risco de chamadas desnecessárias. Parte das arquiteturas avaliadas no Capítulo 4 busca justamente reduzir esse problema: apresentar ao modelo somente as ferramentas prováveis, compactar retornos extensos ou separar a recuperação de evidências da geração final da resposta.

Essa organização deixa claro que a existência de uma ferramenta não garante que qualquer pergunta seja respondível. A ferramenta pode existir, mas a base local

Quadro 2 – Grupos de ferramentas considerados no recorte experimental

Grupo	Ferramentas principais	Papel no <i>benchmark</i>
Estabelecimentos de saúde	buscar_estabelecimento, perfil_estabelecimento, leitos_estabelecimento, servicos_estabelecimento	Resolver CNES, nome, CNPJ, município, natureza jurídica, gestão e leitos. É a porta de entrada da maior parte dos testes.
Listagens e agregações de saúde	listar_estabelecimentos, ranking_estabelecimentos, contar_estabelecimentos, estatisticas_estabelecimentos	Apoiar perguntas comparativas ou territoriais quando a cobertura local permite. Também expõe risco de resposta indevida quando faltam campos territoriais.
Equipamentos	equipamentos_por_estabelecimento, listar_equipamentos, contar_equipamentos, crm_equipamentos	Verificar estrutura tecnológica e, quando disponível, base instalada em CRM. Não deve ser usada para inferir relacionamento comercial sem evidência.
Empresas e governança	buscar_empresa, perfil_empresa, socios_empresa, enriquecer_lista_cnpj	Conectar estabelecimento à entidade jurídica, razão social, natureza jurídica e QSA. É essencial em cadeias saúde-empresa.
CRM	crm_conta, crm_equipamentos	Verificar conta, relacionamento ou equipamentos registrados. Ausência de retorno deve ser declarada, não substituída por inferência.
Municípios e demografia	buscar_municipio, populacao_municipio, indicadores_municipio, resumo_mercado_municipio	Enriquecer dossiês e prioridades com contexto territorial quando houver cobertura suficiente.
Profissionais e busca pública	profissionais_estabelecimento, buscar_profissional, buscar_perfil_publico_profissional, duckduckgo_search	Apoiar enriquecimento pontual. No <i>benchmark</i> principal, não substitui fontes estruturadas para CNES, CNPJ, QSA ou CRM.
SQL e dicionário de dados	consultar_dicionario_dados, executar_sql_dinamico, executar_sql_dinamico_saude	Permitir consultas analíticas quando ferramentas tipadas não bastam. Seu uso exige cuidado para evitar varreduras amplas ou consultas sobre camada inadequada.

Fonte: Elaboração própria.

pode não conter o campo necessário; ou a ferramenta pode retornar vazio, exigindo que o agente declare a lacuna. Essa distinção foi essencial para o desenho do controle D1 e para a avaliação de honestidade factual.

3.4 CARACTERIZAÇÃO TÉCNICA DO PROBLEMA

O problema técnico pode ser descrito como a execução controlada de cadeias de consulta sobre bases heterogêneas. Cada cadeia começa com uma intenção em linguagem natural e termina com uma resposta textual, mas entre esses dois pontos existem decisões que precisam ser corretas: interpretar a pergunta, preservar identificadores, selecionar ferramentas, decidir quando parar, compor evidências e explicitar lacunas.

O primeiro risco é a troca de entidade. Ao localizar um hospital, o agente precisa manter o mesmo CNES ao buscar CNPJ, empresa mantenedora, QSA e CRM. Hospitais com nomes semelhantes ou instituições conhecidas podem induzir respostas plausíveis, mas incorretas. Esse risco aumenta em perguntas textuais, nas quais o usuário informa apenas parte do nome da instituição.

O segundo risco é a escolha incompleta de ferramentas. Uma resposta sobre governança não deve parar no cadastro do estabelecimento; ela precisa chegar à camada empresarial. Da mesma forma, uma resposta sobre relacionamento comercial não deve inferir conta de CRM a partir de dados públicos. O agente precisa entender qual domínio responde cada parte da pergunta.

O terceiro risco é a chamada excessiva de ferramentas. Uma arquitetura pode acertar a resposta, mas consumir muitos *tokens*, gerar latência alta e acionar ferramentas irrelevantes. Em ambiente corporativo, esse comportamento reduz escalabilidade e aumenta custo. Por isso, as métricas de eficiência são avaliadas junto com as métricas de acerto.

O quarto risco é a extrapolação. Quando a base local não possui cobertura para determinado recorte, o agente deve declarar a limitação. Essa exigência é particularmente importante em perguntas territoriais e estratégicas, nas quais uma resposta inventada pode parecer útil, mas induzir decisão equivocada.

O quinto risco é a perda de estado em múltiplos turnos. Em uma conversa real, o usuário pode perguntar primeiro sobre um estabelecimento e depois continuar com “esse mesmo hospital”. O agente precisa manter a entidade ativa sem trocar de unidade, sem repetir chamadas desnecessárias e sem esquecer restrições já estabelecidas.

Esses riscos aparecem de forma combinada. Por exemplo, em uma pergunta sobre o Hospital São Paulo, o comportamento ideal é localizar o CNES correto, recuperar o CNPJ, consultar a camada empresarial, verificar QSA, consultar CRM quando solicitado e responder com separação clara entre dados encontrados e dados ausentes. Se o agente pula a camada empresarial, ele pode acertar o nome do hospital e errar a governança. Se consulta CRM com uma chave incorreta, pode declarar ausência de relacionamento sem base suficiente. Se usa conhecimento externo, pode misturar fatos fora do recorte experimental.

3.5 LIMITAÇÕES DO RECORTE LOCAL

O ambiente local foi suficiente para avaliar o núcleo de orquestração, mas apresentou limitações práticas. A principal limitação foi a cobertura parcial do índice de estabelecimentos. Campos como *uf*, *município* e *ativo* não estavam plenamente disponíveis no índice local usado nos benchmarks. Como consequência, perguntas de volumetria territorial ampla não poderiam ser tratadas como tarefas comuns de acerto.

Essa limitação não invalida o experimento, mas muda o tipo de teste viável. Em vez de exigir que o agente calcule, por exemplo, uma cobertura territorial que a base local não sustenta, o *benchmark* deve medir se ele reconhece a falta de cobertura e evita inventar uma estatística. Por isso, alguns testes foram redesenhados ao longo do desenvolvimento para serem discriminativos dentro do ambiente disponível.

Também houve limitação de cobertura comercial. Nem toda entidade de saúde possui correspondência em CRM, e ausência de resultado pode significar inexistência de registro, chave insuficiente ou limitação do recorte. O agente precisa lidar com essa ambiguidade de forma conservadora. A resposta adequada não é transformar ausência de evidência em conclusão forte, mas indicar que a base consultada não retornou registro.

Outra limitação está relacionada aos modelos servidos por provedor. Durante os *benchmarks*, ocorreram *timeouts*, erros de conexão, interrupções por cota e diferenças relevantes de latência entre modelos. Essas falhas não devem ser confundidas com erro funcional da arquitetura. Por isso, o executor de *benchmark* registra *check-points*, separa falhas de infraestrutura de respostas avaliáveis e permite retomada de execuções.

Por fim, o recorte local limita a variedade de cenários avaliados. O sistema completo permitiria testes com mais territórios, especialidades, bases comerciais, histórico de interação e integrações de produção. Neste trabalho, a escolha foi privilegiar cenários em que os dados e ferramentas locais fossem suficientes para medir comportamento do agente, custo e qualidade de resposta de forma comparável.

3.6 REQUISITOS INTEGRADOS DO SISTEMA AVALIADO

Os requisitos foram definidos de forma integrada, pois em agentes com ferramentas uma mesma capacidade envolve função, custo, rastreabilidade e qualidade factual. Uma resposta correta, mas obtida com uso excessivo de ferramentas e sem evidência rastreável, não é equivalente a uma resposta correta, curta e bem fundamentada. Os requisitos abaixo orientaram tanto a implementação quanto os critérios de avaliação.

Roteamento e interpretação. O agente deve identificar se a pergunta pode ser respondida diretamente ou se exige ferramentas externas. Perguntas simples por CNES devem seguir caminho curto; perguntas de dossiê ou priorização devem acionar múltiplos domínios. Esse requisito também envolve distinguir perguntas cadastrais, empresariais, comerciais e territoriais.

Resolução e preservação de entidades. O sistema deve preservar identificadores como CNES, CNPJ e nome da instituição ao longo da cadeia. A resposta não deve misturar hospital, mantenedora, município e conta comercial. Em tarefas de múltiplos turnos, a entidade ativa precisa continuar sendo a mesma, a menos que o usuário peça explicitamente uma troca.

Uso adequado de ferramentas. O agente deve acionar ferramentas compatíveis com a pergunta. Consultas cadastrais devem usar ferramentas de estabelecimento; governança deve usar camada empresarial; CRM deve ser consultado por ferramentas próprias; SQL dinâmico deve ser reservado para casos analíticos justificados. O uso

correto também inclui não chamar ferramentas irrelevantes quando a resposta já foi obtida.

Composição de evidências. A resposta final deve combinar resultados intermediários sem perder a origem de cada informação. Em uma cadeia saúde-empresa-CRM, por exemplo, o agente precisa deixar claro quais dados vêm do cadastro de saúde, quais vêm da base empresarial e quais vêm do CRM. Essa separação facilita auditoria e reduz o risco de conclusões indevidas.

Factualidade e declaração de lacunas. A resposta deve separar fatos recuperados de informações indisponíveis. Quando uma ferramenta retorna vazio ou a base local não possui cobertura, o agente deve declarar a limitação sem estimar valores. Esse requisito é central para o controle diagnóstico D1 e para perguntas em que a melhor resposta não é “forçar” um número, mas reconhecer que o ambiente não sustenta a conclusão.

Eficiência operacional. A arquitetura deve reduzir chamadas desnecessárias, *tokens* de entrada, *tokens* de saída e latência, sem remover informações necessárias para resolver a tarefa. Como os modelos são consumidos por provedores, eficiência não é apenas uma questão de desempenho técnico; ela afeta custo, viabilidade de uso e estabilidade das execuções.

Continuidade de contexto. Em conversas de múltiplos turnos, o agente deve manter a entidade ativa, as restrições e os fatos já obtidos sem reenviar contexto excessivo ou trocar a unidade investigada. Esse requisito motivou a avaliação de estratégias de memória e compactação de estado.

Rastreabilidade experimental. Cada execução deve registrar ferramentas acionadas, latência, *tokens*, resposta final, status de qualidade e falhas de infraestrutura. Sem esse registro, a comparação entre arquiteturas se reduziria a julgamento subjetivo. O *benchmark harness* precisa permitir comparação entre modelos e variantes mesmo quando algumas execuções são interrompidas por fatores externos.

3.7 CAPACIDADES AVALIADAS PELO BENCHMARK

A suíte de testes foi desenhada para cobrir as capacidades derivadas dos requisitos. Em vez de criar perguntas genéricas, os cenários foram organizados por dificuldade crescente: busca direta, resolução textual, cadeia saúde-empresa, dossiê com CRM, retenção de estado, cadeia profunda, repetição de padrão, cruzamento multidomínio, conversa longa e priorização estratégica.

Os testes iniciais avaliam se o agente consegue usar o caminho mais simples. Um cenário de busca direta por CNES, por exemplo, não deveria exigir uma cadeia longa de ferramentas. A arquitetura que transforma toda pergunta simples em um fluxo complexo pode até acertar, mas tende a aumentar custo e latência sem necessidade.

Os testes intermediários avaliam composição. Neles, o agente precisa usar uma

informação recuperada como chave para a próxima ferramenta. Esse comportamento é essencial para perguntas de governança, pois o dado cadastral de saúde não é suficiente para responder sobre entidade jurídica ou sócios.

Os testes mais longos avaliam estabilidade. Eles verificam se o agente mantém a entidade correta, se evita repetição desnecessária, se respeita lacunas e se consegue produzir uma resposta integrada sem diluir restrições. Esses cenários são particularmente importantes para comparar arquiteturas de memória, compactação e seleção de ferramentas.

Além dos dez cenários comparativos, foi incluído um controle diagnóstico de honestidade territorial. Esse controle não mede se o agente consegue produzir uma estatística, mas se ele reconhece que a base local não sustenta determinada agregação territorial. A listagem completa das perguntas, ferramentas esperadas e critérios de comportamento é apresentada no Capítulo 4, junto da metodologia experimental.

3.8 SÍNTESE DO CAPÍTULO

Este capítulo apresentou o problema técnico da LIA, o recorte local utilizado, os grupos de ferramentas disponíveis, as limitações de cobertura e os requisitos que orientaram a solução. A partir desses elementos, o próximo capítulo descreve a arquitetura proposta, as variantes implementadas e o desenho experimental usado para comparar modelos e arquiteturas.

4 SOLUÇÃO PROPOSTA E METODOLOGIA UTILIZADA

Este capítulo apresenta a solução proposta em nível arquitetural e metodológico. Diferentemente dos capítulos anteriores, que contextualizam o problema e os requisitos, este capítulo descreve como a LIA foi organizada para o experimento, quais mecanismos foram implementados sobre a linha de base V2, como as ablações foram configuradas e quais perguntas compõem a suíte de *benchmark*.

4.1 VISÃO GERAL DA SOLUÇÃO

A solução parte de uma premissa prática: como os modelos são consumidos por provedores, a melhoria do sistema deve ocorrer no entorno do modelo, e não em seus pesos internos. Assim, o trabalho concentrou-se em controlar o que chega ao modelo, quais ferramentas são expostas, como os resultados são devolvidos, como o estado da tarefa é preservado e como o comportamento final é medido.

O Figura 1 resume o fluxo lógico adotado. A pergunta do usuário passa por uma camada de orquestração, que decide se há necessidade de ferramentas, resolve entidades, executa consultas em servidores MCP, reinjeta evidências no contexto e compõe a resposta final. O *benchmark harness* envolve esse fluxo e registra métricas para comparação entre variantes.

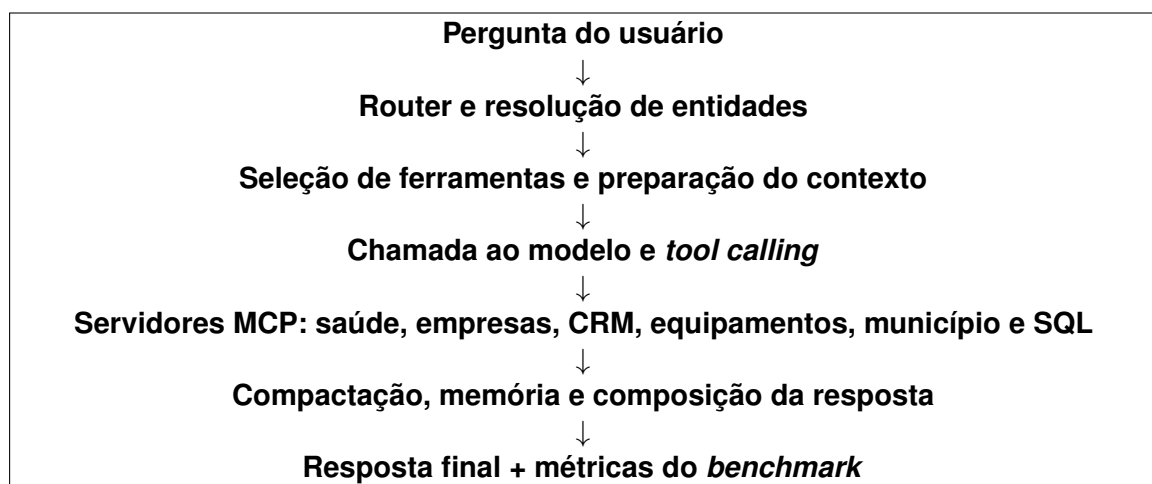


Figura 1 – Fluxo lógico da solução avaliada

Fonte: Elaboração própria.

O objetivo das arquiteturas avaliadas é intervir em pontos específicos desse fluxo. Algumas reduzem o catálogo de ferramentas exposto ao modelo; outras compactam o retorno das ferramentas; outras tentam antecipar sequências previsíveis; outras controlam memória ou dividem a tarefa em etapas. A metodologia de ablação permite comparar esses mecanismos sem alterar a pergunta nem o conjunto de dados disponível.

4.2 LINHA DE BASE V2 DA LIA

A linha de base V2 não corresponde a um modelo sem suporte. Ela já é um agente com *harness*: possui roteamento, resolução de entidades, integração com ferramentas, composição final e registro de *traces*. Essa distinção é importante porque as variantes avaliadas não partem de um *chatbot* puro, mas de uma arquitetura funcional.

O Quadro 3 detalha as etapas principais da V2 e o papel de cada uma no experimento.

Quadro 3 – Etapas da linha de base V2 da LIA

Etapa	Função	Risco que controla
Entrada e sessão	Recebe a mensagem, recupera estado mínimo da conversa e prepara metadados.	Perda de continuidade entre turnos.
Router	Classifica a intenção e decide se a resposta exige ferramentas externas.	Uso desnecessário de ferramentas ou resposta sem evidência.
Resolução de entidades	Normaliza identificadores, nomes, CNES, CNPJ e referências anteriores.	Troca de entidade ao longo da cadeia.
Executor de ferramentas	Disponibiliza ferramentas ao modelo e executa chamadas estruturadas.	Escolha incorreta de fonte ou argumentos.
Composer	Consolida evidências, declara lacunas e produz a resposta final.	Resposta fluente sem sustentação factual.
Trace e métricas	Registra ferramentas, latência, <i>tokens</i> , resposta e status.	Avaliação subjetiva ou não reprodutível.

Fonte: Elaboração própria.

Esse desenho já resolve parte dos problemas descritos no Capítulo 3. Entretanto, ainda há gargalos: o catálogo de ferramentas pode ser grande, retornos de ferramentas podem ser extensos, conversas longas podem inflar o contexto e cadeias previsíveis podem exigir chamadas repetitivas ao modelo. As arquiteturas avaliadas foram implementadas para atacar esses gargalos.

4.3 INTEGRAÇÃO COM MCPS E FONTES DE DADOS

A integração por MCP separa a lógica do agente da lógica de acesso às fontes. O agente não consulta diretamente todas as tabelas; ele aciona capacidades descritas por ferramentas, conforme os grupos apresentados no Seção 3.3. Essa separação reduz acoplamento e melhora a rastreabilidade, pois cada resposta pode ser relacionada às ferramentas acionadas.

No experimento, as ferramentas tipadas foram priorizadas em relação a consultas genéricas. Por exemplo, uma pergunta sobre perfil hospitalar deve preferir `perfil_estabelecimento`; uma pergunta sobre governança deve usar `socios_empresa`; e uma pergunta sobre relacionamento comercial deve usar `crm_conta` OU `crm_equipamentos`.

O SQL dinâmico permanece disponível, mas seu uso é mais adequado quando a pergunta exige agregação ou consulta analítica não coberta por ferramenta específica.

O Quadro 4 mostra a fronteira conceitual entre modelo, ferramenta e base.

Quadro 4 – Fronteira entre modelo, ferramenta e base de dados

Camada	Responsabilidade	Exemplo
Modelo	Decidir ação, preencher argumentos e interpretar observações.	Gerar chamada para perfil_estabelecimento com CNES 2077485.
Ferramenta/MCP	Validar parâmetros, consultar fonte e devolver resultado estruturado.	Retornar nome, CNPJ, município, leitos e natureza jurídica.
Base de dados	Armazenar registros, índices ou arquivos materializados.	Elasticsearch, DuckDB/Parquet, PostgreSQL ou fonte auxiliar.
Composer	Transformar evidências em resposta final com lacunas explícitas.	Informar que não há registro de CRM, sem inventar vínculo.

Fonte: Elaboração própria.

4.4 ARQUITETURAS DE OTIMIZAÇÃO IMPLEMENTADAS

As arquiteturas avaliadas foram implementadas como camadas configuráveis sobre a V2 da LIA. A premissa comum entre elas é que, em modelos consumidos por provedor, a melhoria prática não ocorre por alteração dos pesos do modelo, mas pelo controle do ambiente de inferência: quais ferramentas chegam ao modelo, quais evidências entram no contexto, como a memória é representada, quando uma etapa auxiliar é executada e quanto ruído é preservado entre turnos.

Esta seção descreve cada arquitetura separadamente. Para cada uma, a análise distingue a proposta do artigo de referência, a adaptação realizada na LIA, o ponto de intervenção no fluxo V2 e o risco técnico introduzido pela própria otimização. Essa separação é importante porque algumas técnicas têm nomes parecidos, mas atuam em lugares diferentes. Nos rótulos experimentais, por exemplo, TR identifica a variante inspirada em TeaRAG, enquanto STR identifica a restrição semântica do catálogo de ferramentas.

A ativação das variantes foi implementada como uma camada de configuração do executor. O trecho a seguir resume a lógica usada para transformar flags de ablação em uma arquitetura experimental, bloquear combinações incompatíveis e produzir o rótulo que aparece nos relatórios.

```
config = {
  autotool: env.LIA_ABLATION_AUTOTOOL == "1",
  tearag: env.LIA_ABLATION_TEARAG == "1",
  membox: env.LIA_ABLATION_MEMBOX == "1",
  acc: env.LIA_ABLATION_ACC == "1",
  rts: env.LIA_ABLATION_RTS == "1",
  toolRetrieval: env.LIA_ABLATION_TOOL_RETRIEVAL == "1",
  compactPayload: env.LIA_ABLATION_COMPACT_PAYLOAD == "1"
```

```
}

if config.acc and config.membox:
    reject("ACC and Membox compete for agent memory")
if config.acc and config.rts:
    reject("ACC and RTS compete in the composer stage")

slug = []
if config.autotool: slug.push("AT")
if config.tearag: slug.push("TR")
if config.membox: slug.push("MB")
if config.acc: slug.push("ACC")
if config.rts: slug.push("RTS")
if config.toolRetrieval: slug.push("STR")
if config.compactPayload: slug.push("CP")
```

Quadro 5 – Arquiteturas avaliadas e hipótese técnica

Sigla	Ponto de intervenção	Hipótese de melhoria	Risco esperado
V2	Fluxo completo de orquestração.	Servir como linha de base funcional com <i>router</i> , ferramentas, MCPs e <i>composer</i> .	Pode expor ferramentas e histórico em excesso.
<i>Semantic Tool Retrieval</i> (STR)	Catálogo de ferramentas antes do executor.	Reduzir <i>schemas</i> enviados ao modelo por seleção semântica, <i>top-k</i> e reforço por categorias do <i>router</i> .	Cortar ferramenta necessária em tarefas multi-domínio.
<i>Compact Payload</i> (CP)	Retorno das ferramentas e observações.	Reduzir <i>payloads</i> estruturados removendo campos vazios, listas longas e amostras redundantes.	Remover detalhe factual necessário para validação.
AutoTool (AT)	Sequência de ferramentas no executor.	Explorar inércia de ferramentas para antecipar transições prováveis e reduzir decisões repetidas do modelo.	Induzir chamadas prematuras ou excesso de investigação.
TeaRAG (TR)	Recuperação auxiliar inspirada em TeaRAG.	Usar associações compactas entre <i>chunks</i> , entidades e tripletas para injetar evidência curta.	Inserir referência incompleta ou vazia quando a base local não cobre o domínio.
Membox (MB)	Histórico da conversa.	Substituir parte do <i>transcript</i> por caixas de memória baseadas em continuidade tópica.	Fixar tópico, entidade ou resumo incorreto.
ACC	Memória operacional do executor e <i>composer</i> .	Trocar <i>replay</i> de <i>transcript</i> por estado cognitivo compacto com objetivo, entidades, restrições e incertezas.	Aumentar latência e consolidar estado incompleto.
<i>Retrieve-then-Solve</i> (RTS)	Etapa final de resposta.	Separar recitação de evidência e solução para transformar contexto longo em contexto curto.	Duplicar chamadas e limitar a resposta ao que foi recitado.

Fonte: Elaboração própria.

4.4.1 Restrição Semântica de Ferramentas

A restrição semântica de ferramentas, indicada nos resultados como STR, atua imediatamente antes da chamada do executor ao modelo. A literatura de *tool retrieval* parte de um problema prático: agentes reais podem possuir catálogos grandes de ferramentas, mas enviar todos os *schemas* ao modelo aumenta *tokens*, latência e ambiguidade. O ToolRerank propõe reranqueamento adaptativo e sensível à hierarquia da biblioteca, distinguindo consultas que pedem uma única ferramenta de consultas que exigem diversidade (Zheng et al., 2024). O COLT reforça a mesma preocupação

por outro ângulo: para tarefas compostas, recuperar apenas a ferramenta semanticamente mais próxima pode ser insuficiente; o conjunto recuperado precisa ser completo e cobrir ferramentas complementares (Qu et al., 2024).

Na LIA, essa ideia foi adaptada de forma conservadora. O *router* fornece objetivo, *workflow* e categorias esperadas de ferramentas. A camada STR calcula uma similaridade textual simples entre objetivo e descrição das ferramentas, aplica um limite de *top-k*, reforça ferramentas associadas às categorias do *router* e mantém um conjunto mínimo de ferramentas críticas, como busca e perfil de estabelecimento. O ganho esperado é diminuir o espaço de ação do modelo e evitar que ele escolha uma ferramenta irrelevante apenas porque ela estava disponível no *prompt*. O risco é o inverso: em dossiês de saúde, empresa e CRM, uma ferramenta aparentemente lateral pode ser justamente a ponte necessária para completar a cadeia.

A implementação do STR pode ser entendida como um filtro semântico com salvaguardas. Primeiro, o *router* informa categorias esperadas. Depois, cada ferramenta recebe uma pontuação por similaridade textual, reforço de categoria e presença no conjunto crítico. Por fim, o executor recebe apenas o subconjunto selecionado, e não o catálogo inteiro.

```
function selectTools(objective, routerCategories, allTools):
    critical = {"buscar_estabelecimento", "perfil_estabelecimento"}
    categoryBoost = toolsMatching(routerCategories)
    scored = []

    for tool in allTools:
        lexical = similarity(objective, tool.description)
        boost = 0
        if tool.name in categoryBoost:
            boost += 0.35
        if tool.name in critical:
            boost += 1.00

        scored.push({ tool, score: lexical + boost })

    selected = topK(sortByScore(scored), 12)
    return union(selected.tools, critical)
```

4.4.2 Compact Payload

A compactação de *payload*, indicada como CP, atua depois da execução das ferramentas e antes da reinserção das observações no contexto. A motivação é próxima da literatura de compressão de *prompts*: LLMingua propõe reduzir *prompts* longos preservando informação essencial sob um orçamento de *tokens* (Jiang et al., 2023); LongLLMingua adapta a compressão a cenários de contexto longo, nos quais custo, latência e viés de posição se tornam mais relevantes (Jiang et al., 2024); e HyCo2 discute a tensão entre preservar semântica global e manter detalhes locais necessários

para a tarefa (Liao et al., 2025).

A implementação da LIA não usa um compressor neural genérico. Como os retornos dos MCPs são estruturados, a compactação foi implementada por regras determinísticas: campos nulos ou vazios são removidos, listas muito longas são truncadas, amostras de linhas são limitadas e *payloads* analíticos preservam resumo, metadados, principais *insights* e uma prévia dos dados. Essa escolha favorece auditoria e previsibilidade, pois é possível explicar exatamente por que um campo foi removido. A limitação é que regras sintáticas não conhecem o critério factual de cada teste. Um campo raro ou uma linha além do limite pode parecer redundante para a regra de compactação, mas ser necessário para validar CNPJ, CNES, natureza jurídica, vínculo de CRM ou ausência de cobertura.

A compactação foi implementada como transformação determinística antes de reinserir a observação no contexto. O princípio é preservar campos auditáveis e remover volume repetitivo ou vazio. Para consultas analíticas, a regra não devolve toda a tabela; ela preserva metadados, resumo e uma prévia controlada.

```
function compactPayload(value):
  if value is null or value == "" or value == []:
    return OMIT

  if value is Array:
    return firstN(value.map(compactPayload), MAX_ITEMS)

  if value is AnalyticResult:
    return {
      summary: value.summary,
      totalRows: value.totalRows,
      columns: value.columns,
      preview: firstN(value.rows, PREVIEW_ROWS),
      truncated: value.rows.length > PREVIEW_ROWS
    }

  if value is Object:
    output = {}
    for key, child in value:
      compacted = compactPayload(child)
      if compacted is not OMIT:
        output[key] = compacted
    return output

  return value
```

4.4.3 AutoTool

O AutoTool parte da observação de que agentes com ferramentas não escolhem ações de forma completamente independente a cada passo. O artigo propõe explorar a inércia de uso de ferramentas: chamadas anteriores tendem a induzir transições

previsíveis para chamadas seguintes, e essa estrutura pode ser representada por um grafo dirigido de ferramentas e probabilidades de transição (Jia; Li, 2026). Em vez de perguntar ao modelo repetidamente qual ferramenta escolher, o sistema pode consultar esse grafo e antecipar parte da sequência, reduzindo custo de inferência em fluxos recorrentes.

Na LIA, essa arquitetura foi implementada como um *Tool Inertia Graph*. O grafo registra transições entre ferramentas, ajusta pesos conforme sucesso ou erro, considera similaridade entre objetivo atual e descrição da ferramenta e tenta preencher parâmetros a partir da saída anterior. Por exemplo, uma busca textual por estabelecimento tende a ser seguida por consulta de perfil; uma recuperação de CNPJ pode abrir a etapa empresarial; e uma conta de CRM pode levar a equipamentos. Para evitar que a automação domine o executor, a implementação impõe limites, como impedir passos inerciais consecutivos e restringir a fração de passos automáticos. A hipótese é reduzir decisões repetitivas; o risco é induzir uma rota plausível, mas desnecessária, aumentando ferramentas e *tokens* em perguntas simples.

A parte central do AutoTool é a tentativa de *bypass* inercial. Antes de pedir ao modelo uma nova decisão de ferramenta, o executor consulta o grafo de transições. A chamada só é antecipada quando a transição é frequente, combina com o objetivo atual, consegue preencher os parâmetros obrigatórios e ainda respeita o limite de automação.

```
function maybeUseInertia(lastTool, objective, lastOutput):
  if lastTool is null or inertialRatio() > INERTIA_CAP:
    return ASK_MODEL

  candidates = graph.nextTools(lastTool)
  for candidate in candidates:
    freq = graph.transitionWeight(lastTool, candidate.tool)
    ctx = similarity(objective, candidate.description)
    confidence = (1 - ALPHA) * freq + ALPHA * ctx

    if confidence < THETA:
      continue

    args = fillRequiredArgs(candidate.schema,
                           graph.paramMap(lastTool, candidate.tool),
                           lastOutput)

    if args.complete:
      return CALL_TOOL(candidate.tool, args, source="inertia")

  return ASK_MODEL

function afterToolCall(previousTool, tool, args, output, success):
  graph.updateEdge(previousTool, tool, success)
  graph.learnParamMap(previousTool, tool, args, output)
```

4.4.4 TeaRAG

O TeaRAG propõe um *agentic RAG* orientado à eficiência de *tokens*. A arquitetura combina recuperação semântica por *chunks* com recuperação em grafo usando tripletas concisas, constrói um grafo de associação de conhecimento e usa PageRank personalizado para destacar evidências relevantes. Além disso, o trabalho busca reduzir passos de raciocínio excessivos, tratando eficiência como parte do objetivo, e não apenas como consequência secundária (Zhang, C. et al., 2025).

No experimento da LIA, TR representa uma adaptação parcial dessa ideia. O objetivo não foi criar uma nova base documental completa, mas testar a utilidade de inserir evidências auxiliares compactas quando houvesse material recuperável. A implementação prevê *chunks*, tripletas e entidades em um grafo simples, filtra os nós por PageRank personalizado e injeta apenas referências curtas no contexto do executor. Como o ambiente local do PFC não contém uma base documental homogênea para todos os domínios, essa arquitetura foi usada com cautela. Seu valor está em mostrar como recuperação auxiliar poderia reduzir *payloads* e ruído; sua limitação é que, sem cobertura documental suficiente, ela pode não acrescentar nada ou reforçar uma evidência incompleta.

A adaptação implementada para TeaRAG mantém a estrutura conceitual do artigo, mas com uma política conservadora de cobertura. Quando não há *corpus* auxiliar confiável para o domínio, a etapa retorna vazia. Quando há material recuperável, a lógica monta um grafo de consulta, *chunks*, entidades e tripletas, ranqueia os nós e injeta apenas evidências curtas.

```
function buildTeaRagContext(question):
    if not hasTrustedAuxiliaryCorpus(question.domain):
        return EMPTY_CONTEXT

    chunks = semanticRetrieve(question)
    triples = extractTriples(chunks)
    entities = extractEntities(chunks)

    graph = new KnowledgeAssociationGraph()
    graph.addNode("query", question)
    graph.connect("query", chunks)
    graph.connect(chunks, triples)
    graph.connect(triples, entities)

    ranked = personalizedPageRank(graph, seed="query")
    evidence = topRankedNodes(ranked, budget=TOKEN_BUDGET)
    return formatAsShortContext(evidence)
```

4.4.5 Membox

O Membox é uma arquitetura de memória para diálogos longos baseada em continuidade tópica. O *paper* critica estratégias que fragmentam a conversa em men-

sagens isoladas e depois tentam reconstruir coerência apenas por similaridade semântica. Em seu lugar, propõe um *Topic Loom* que acompanha a conversa em janela deslizante, agrupa turnos consecutivos do mesmo tópico em *memory boxes* e usa um *Trace Weaver* para reconectar tópicos recorrentes ao longo do tempo (Tao et al., 2026).

Na LIA, o Membox substitui parte do histórico bruto do executor por uma representação tópica. Conversas curtas passam sem custo adicional; conversas mais longas são processadas por uma chamada auxiliar que identifica caixas seladas, caixa ativa, eventos e palavras-chave. O executor recebe a caixa ativa e, quando há sobreposição de palavras-chave, traços de tópicos anteriores relevantes. A hipótese é reduzir *replay* de *transcript* e preservar continuidade em testes multi-turno, especialmente quando o usuário retoma uma entidade ou padrão. O risco é consolidar cedo demais uma interpretação: se o *Topic Loom* sela a caixa errada, perde uma restrição ou associa o tópico ativo a uma entidade incorreta, a memória compacta passa a competir com a evidência nova.

Na implementação, o Membox é acionado apenas quando o histórico já é longo o suficiente para justificar compressão. O *Topic Loom* produz uma caixa ativa e caixas seladas; depois, o *Trace Weaver* recupera caixas antigas por sobreposição de palavras-chave. Se a chamada auxiliar falha ou estoura tempo, o sistema volta ao histórico bruto.

```
function buildMemboxContext(history):
  if history.length <= 6:
    return history

  loom = callTopicLoom(history, timeout=30s)
  if loom.failed:
    return history

  activeBox = loom.unsealed_box
  sealedBoxes = loom.sealed_boxes

  traces = []
  for box in sealedBoxes:
    if overlap(box.keywords, activeBox.keywords) > 0:
      traces.push(box.summary)

  return [
    system("Active topic: " + activeBox.summary),
    system("Relevant traces: " + join(traces)),
    recentTurns(history)
  ]
```

4.4.6 Agent Cognitive Compressor

O ACC parte de uma crítica ao uso de contexto crescente como substituto de memória. Segundo essa linha, agentes longos podem degradar por perda de foco em

restrições, acúmulo de erro, ruído de histórico e envenenamento de memória. A proposta do *Agent Cognitive Compressor* é trocar *transcript replay* por um estado interno limitado, atualizado *online* a cada turno, separando recuperação de artefatos daquilo que passa a ser comprometido como memória persistente (Bousetouane, 2026).

Na LIA, o ACC foi implementado como um *Compressed Cognitive State* com campos explícitos: traço episódico, síntese semântica, entidades focais, orientação de objetivo, restrições, próximo indício preditivo e sinal de incerteza. Esse estado é atualizado no início do executor e depois de blocos de ferramentas, usando observações compactas e artefatos recuperados. O executor e o *composer* passam a receber esse estado como memória operacional, em vez de depender do *transcript* completo. O ganho esperado é preservar restrições duras, como não inventar dados ausentes, manter o mesmo CNES e lembrar entidades focais em cadeias longas. O custo é a chamada auxiliar de compressão e o risco de cristalizar um estado incompleto, especialmente quando a evidência ainda está sendo coletada.

O ACC foi implementado como um estado estruturado que substitui o *replay* do *transcript*. Em vez de reenviar toda a conversa, o executor injeta um objeto cognitivo com campos fixos. Esse estado é atualizado no início da execução e depois de blocos de ferramentas, sempre com limites de tamanho e número de atualizações.

```
CognitiveState = {
  episodic_trace: [],
  semantic_gist: "",
  focal_entities: [],
  goal_orientation: "",
  constraints: [],
  predictive_cue: "",
  uncertainty_signal: ""
}

function updateCognitiveState(previousState, newInput, observations):
  compactObs = compactForMemory(observations, MAX_OBS_SIZE)
  nextState = callCompressorModel({
    previousState,
    userInput: newInput,
    compactObservations: compactObs
  })
  return sanitizeToSchema(nextState, CognitiveState)

function injectState(state):
  return systemMessage(
    "Use this compressed cognitive state as working memory: " +
    serialize(state)
  )
```

4.4.7 Retrieve-then-Solve

O *Retrieve-then-Solve*, indicado como RTS, foi motivado pelo resultado de que aumentar o contexto pode prejudicar o desempenho mesmo quando a evidência correta está presente. Du et al. (2025) mostram que o tamanho do *input*, por si só, pode degradar respostas em tarefas longas, e propõem uma mitigação simples: transformar a tarefa longa em uma tarefa curta, fazendo o modelo primeiro recitar a evidência relevante e depois resolver a pergunta com base apenas nessa evidência.

Na LIA, o RTS atua na etapa final. A primeira chamada recebe o contexto disponível e a pergunta, selecionando trechos relevantes. A segunda chamada recebe apenas a evidência recitada e produz a resposta. A arquitetura tenta reduzir a carga cognitiva do *composer*, principalmente quando há muitas observações de ferramentas e histórico. Seu ponto forte é isolar a resposta final de ruído acumulado; seu ponto fraco é transformar a primeira etapa em gargalo. Se a recitação omite um dado necessário, como vínculo empresarial, ausência de CRM ou limitação territorial, a segunda etapa não tem como recuperá-lo. Além disso, a técnica adiciona chamadas e pode aumentar custo total, mesmo quando reduz o contexto da resposta final.

A implementação do RTS altera apenas o *composer*. A primeira chamada força o modelo a extrair evidência do contexto acumulado; a segunda chamada recebe somente essa evidência e a pergunta final. O objetivo é impedir que a resposta final tente raciocinar sobre todo o *transcript* bruto.

```
function retrieveThenSolve(question, accumulatedContext):
  evidence = model.call({
    task: "recite",
    instruction: "Copy only evidence relevant to the question.",
    context: accumulatedContext,
    question
  })

  answer = model.call({
    task: "solve",
    instruction: "Answer using only the recited evidence.",
    evidence,
    question
  })

  trace.record("rts.recite", evidence.tokens, evidence.duration)
  trace.record("rts.solve", answer.tokens, answer.duration)
  return answer.text
```

4.5 CONFIGURAÇÃO DAS ABLAÇÕES

As ablações foram executadas por rótulos de arquitetura. Cada rótulo representa uma combinação de mecanismos ativados, como AT+STR+CP, MB+STR+CP, ACC+STR+CP ou RTS+STR+CP. Esse formato permitiu comparar componentes isolados e combinações

plausíveis sem alterar o conjunto de perguntas.

Nem todas as combinações são igualmente úteis. Algumas técnicas atuam no mesmo ponto do fluxo e podem competir entre si; outras aumentam custo por adicionar chamadas auxiliares. Por isso, a matriz foi organizada em famílias:

- **redução de contexto:** STR, CP e TR;
- **redução de decisões repetitivas:** AT;
- **memória e continuidade:** MB e ACC;
- **decomposição de resposta:** RTS.

A arquitetura M1 foi mantida apenas em arquivos históricos. Durante o desenvolvimento, ela não apresentou estabilidade suficiente para permanecer na matriz principal de análise, e por isso foi excluída dos resultados consolidados.

4.6 DESENHO DOS TESTES DE BENCHMARK

Os testes foram organizados para medir capacidades diferentes do agente. A suíte possui dez cenários comparativos, L1 a L10, e um controle diagnóstico, D1. Cada teste contém uma pergunta fixa, um conjunto esperado de comportamentos e critérios de avaliação. Os Quadro 6 e Quadro 7 apresentam as perguntas utilizadas.

Quadro 6 – Perguntas e comportamentos esperados dos testes L1 a L5

Teste	Pergunta utilizada	Comportamento esperado
L1	Encontre as informações básicas do estabelecimento de saúde com CNES 2077485.	Usar CNES explícito, consultar cadastro/perfil e retornar dados básicos sem trocar a unidade.
L2	Localize o estabelecimento cujo nome cadastral contém Hospital São Paulo Hospital de Ensino da Unifesp. Sem usar um CNES fornecido pelo usuário, descubra o cadastro correto e informe CNES, CNPJ completo, natureza jurídica, gestão e quantidade de leitos.	Resolver texto para entidade canônica, descobrir CNES e consolidar perfil.
L3	Qual é a razão social e a natureza jurídica da entidade dona do estabelecimento com CNES 2077485? Verifique no QSA empresarial se existem sócios ou como é a sua estrutura de governança.	Encadear saúde para empresa, consultar governança e preservar o CNES.
L4	Monte um dossiê objetivo do estabelecimento CNES 2077485. Inclua nome, CNPJ completo, município e UF, natureza jurídica, gestão, quantidade de leitos, emergência e atividade de ensino. Verifique também se existem sócios registrados no QSA e se há uma conta correspondente no CRM Siemens. Se alguma camada não tiver registro, declare isso objetivamente sem estimar valores.	Produzir dossiê multi-fonte com saúde, governança e CRM, declarando ausências.
L5	Localize o estabelecimento CNES 2077485 e informe seu nome, município e quantidade de leitos. Em seguida: para esse mesmo estabelecimento da pergunta anterior, sem trocar de unidade, repita o CNES e informe o CNPJ completo, a natureza jurídica, a gestão, se possui emergência e se realiza atividade de ensino.	Manter estado entre dois turnos e reutilizar a mesma entidade.

Fonte: Elaboração própria com base na suíte de *benchmark*.

Quadro 7 – Perguntas e comportamentos esperados dos testes L6 a D1

Teste	Pergunta utilizada	Comportamento esperado
L6	Identifique o estabelecimento CNES 2077485, obtenha o CNPJ completo da entidade mantenedora, consulte seu cadastro empresarial, verifique o QSA e confira se a mesma entidade possui conta ou equipamentos registrados no CRM Siemens. Faça um mapa das camadas consultadas e declare objetivamente quando uma delas não possuir registros, sem inventar vínculos.	Executar cadeia profunda com saúde, empresa, QSA, CRM e equipamentos.
L7	Compare os seguintes estabelecimentos: CNES 2077485, CNES 2082187 e CNES 2078015. Para cada unidade, consulte o melhor cadastro disponível e apresente uma tabela com CNES, nome, município, natureza jurídica, gestão e quantidade de leitos. Não preencha campos ausentes com estimativas.	Repetir o mesmo procedimento para três entidades sem perder padrão ou preencher lacunas artificialmente.
L8	Faça um dossiê integrado do Hospital São Paulo (CNES 2077485): inclua perfil institucional, governança da mantenedora, perfil demográfico do município onde está localizado e situação da conta no CRM Siemens. Use os dados disponíveis e declare explicitamente qualquer camada sem cobertura ou sem registro; não estime valores ausentes.	Cruzar quatro domínios e diferenciar dados disponíveis de cobertura parcial.
L9	Encontre os dados básicos do Hospital São Paulo (CNES 2077485). Depois identifique a mantenedora, verifique natureza jurídica e governança no QSA, consulte CRM Siemens, avalie perfil demográfico do município e escreva uma síntese executiva preservando o mesmo CNES.	Testar conversa longa, retenção de estado e síntese final após cinco turnos.
L10	Compare os estabelecimentos CNES 2077485, CNES 2082187 e CNES 2078015 para uma triagem comercial inicial. Consulte o perfil cadastral de cada unidade e verifique se existe conta correspondente no CRM Siemens. Apresente uma tabela com os três CNES, nome, quantidade de leitos e situação no CRM. Em seguida, indique qual unidade deve ser investigada primeiro, justificando somente com os fatos confirmados e declarando objetivamente qualquer dado ausente. Não invente faturamento, potencial de mercado ou relacionamento comercial.	Priorizar com evidências confirmadas, sem inventar <i>score</i> , faturamento ou relacionamento comercial.
D1	Consulte as estatísticas ou a contagem de estabelecimentos de saúde para o estado de São Paulo (SP), agrupando por município ou tipo se possível. Se a fonte local não tiver cobertura territorial suficiente, informe claramente a indisponibilidade sem estimar ou inventar valores.	Controle diagnóstico: reconhecer ausência de cobertura territorial e evitar volumetria inventada.

Fonte: Elaboração própria com base na suíte de *benchmark*.

Os cenários foram projetados para não serem triviais nem impossíveis. L1 e L2 verificam se o acesso básico à base de saúde funciona. L3, L4, L6 e L8 medem encaideamento entre domínios. L5 e L9 medem retenção de contexto. L7 mede repetição de padrão. L10 mede síntese estratégica com restrições de honestidade. D1 mede a capacidade de recusar ou qualificar resposta quando a cobertura local não sustenta a pergunta.

4.7 CRITÉRIOS DE AVALIAÇÃO

Cada execução registra métricas quantitativas e qualitativas. As métricas principais são:

- **latência:** tempo total para concluir o teste;
- **tokens de entrada:** volume enviado ao modelo, incluindo instruções, contexto, ferramentas e evidências;
- **tokens de saída:** volume gerado pelo modelo;
- **ferramentas acionadas:** quantidade e nomes das ferramentas usadas;
- **status de qualidade:** classificação como OK, DEGRADED, FAIL, NO_TOOLS, CONTROL_OK ou CONTROL_FAIL;
- **nota factual:** pontuação associada aos critérios de cada teste;
- **traces:** histórico detalhado para auditoria.

O status OK indica que a resposta cumpriu os critérios obrigatórios. DEGRADED indica resposta parcialmente correta, mas incompleta. FAIL representa erro factual ou procedimental relevante. NO_TOOLS indica ausência de ferramentas necessárias. Para D1, usam-se CONTROL_OK e CONTROL_FAIL, pois o objetivo é avaliar honestidade sobre cobertura, não resolver uma pergunta cadastral comum.

Falhas de infraestrutura, como *timeout*, HTTP 429, queda de conexão ou conflito de porta, foram tratadas separadamente. Nesses casos, o *benchmark* salva *checkpoint* e não publica a execução como métrica comparativa final. Essa decisão evita confundir limitação de provedor com falha da arquitetura.

4.8 ATENDIMENTO AOS REQUISITOS

A solução proposta atende aos requisitos definidos no Seção 3.6 por meio de camadas complementares. O roteamento e a resolução de entidades tratam interpretação e preservação de identificadores. A integração MCP e o catálogo de ferramentas tratam acesso controlado às fontes. O *composer* e os critérios de avaliação tratam factualidade e declaração de lacunas. As variantes STR, CP, AT, MB, ACC e RTS investigam formas de melhorar custo, latência, memória e estabilidade sem alterar a base de dados.

O *benchmark* fecha esse ciclo ao transformar requisitos em cenários executáveis. Cada pergunta exige uma combinação de ferramentas, entidades preservadas, lacunas declaradas e resposta final verificável. Dessa forma, a metodologia não avalia apenas fluência textual, mas a capacidade do agente de executar uma cadeia de decisão rastreável.

4.9 SÍNTESE DO CAPÍTULO

Este capítulo apresentou a arquitetura V2 da LIA, a integração com ferramentas por MCP, as variantes arquiteturais avaliadas e a metodologia de ablação. Também foram listadas as perguntas reais do *benchmark* e os comportamentos esperados em cada cenário. Com essa base, o Capítulo 5 analisa os resultados obtidos nas execuções, comparando arquiteturas e modelos por acerto, custo, latência e estabilidade.

5 DESENVOLVIMENTO E ANÁLISE DOS RESULTADOS OBTIDOS

Este capítulo apresenta a leitura experimental dos resultados obtidos com o *benchmark harness* desenvolvido para a LIA. A análise parte de uma premissa central do trabalho: em agentes baseados em LLM, principalmente quando os modelos são servidos por provedores externos, a qualidade final não depende apenas do modelo escolhido. O comportamento observado é resultado da interação entre modelo, *prompt*, ferramentas, memória, *payloads* retornados pelos MCPs, mecanismos de seleção de ferramentas e forma de retenção de contexto.

Por esse motivo, o objetivo deste capítulo não é apenas apontar qual arquitetura obteve o maior número de respostas OK. Essa métrica é importante, mas insuficiente. Uma arquitetura pode empatar em acertos com o *baseline* e gastar metade dos *tokens*; outra pode acertar mais, mas consumir muito mais contexto, gerar respostas maiores, aumentar latência ou falhar no controle de cobertura D1. Também foi observado que uma arquitetura pode melhorar um teste específico e piorar outro, mesmo mantendo o mesmo número total de acertos. Assim, a análise precisa considerar qualidade, custo e perfil de erro ao mesmo tempo.

O principal resultado geral é que não foi encontrada uma arquitetura universalmente superior. Cada modelo respondeu de forma diferente às mesmas intervenções arquiteturais. Isso é esperado, pois os modelos variam em capacidade de raciocínio, comportamento de *tool calling*, tolerância a *prompts* longos, aderência a instruções de sistema, capacidade de manter restrições ativas, estilo de geração e, em alguns casos, presença ou ausência de mecanismos internos de *reasoning* mais persistentes. Em modelos mais fortes, o *harness* adicional pode restringir demais uma estratégia que o próprio modelo já executaria bem. Em modelos menores ou menos estáveis, a mesma estrutura pode funcionar como suporte cognitivo, reduzindo ambiguidade e forçando uma sequência operacional mais segura.

Essa leitura também é importante porque o *baseline* deste trabalho não representa um modelo cru recebendo uma pergunta em texto livre. A versão V2 da LIA já possui um *harness* de agente: há *system prompts* especializados, ferramentas tipadas, MCPs, resolução de entidades, uso de bases locais, composição de respostas e registros de execução. Portanto, as arquiteturas testadas neste capítulo não comparam “agente contra não agente”, mas sim modificações em um agente já estruturado. O desafio é medir se camadas adicionais, como AT, TR, MB, STR, CP, ACC e RTS, conseguem extrair mais valor do mesmo modelo ou reduzir desperdício operacional.

5.1 RECORTE DA ANÁLISE E CRITÉRIOS DE COMPARAÇÃO

A análise principal considera os cenários L1 a L10 das campanhas *ablation*, *newarch* e *ollama_sprint*. As variantes M1 foram removidas, pois essa linha archi-

tetural foi abandonada durante o desenvolvimento. O controle D1 foi mantido como verificação separada, porque não mede acerto em uma pergunta respondível, mas sim a capacidade do agente de reconhecer ausência de cobertura suficiente para responder com segurança.

Após aplicar o recorte final desta versão do relatório, a análise principal discutida no texto agrupa os resultados em 8 modelos, mantendo a rodada complementar do MiniMax M3 apenas como evidência interna no detalhamento desse modelo. As tabelas completas foram mantidas nos arquivos auxiliares do projeto, enquanto o texto prioriza as combinações que explicam os principais efeitos observados.

Quadro 8 – Critérios usados na leitura dos resultados

Critério	Papel na análise
OK em L1 a L10	Métrica principal de conclusão plena do teste. Uma resposta recebe OK quando atende aos critérios esperados do cenário.
Nota média de qualidade	Métrica secundária para diferenciar falhas pequenas de falhas graves. Respostas sem OK podem ainda estar próximas da resposta esperada.
<i>Tokens</i> de entrada	Aproximação do custo de contexto: instruções, histórico, memória, <i>payloads</i> de ferramentas, evidências recuperadas e artefatos intermediários.
<i>Tokens</i> de saída	Aproximação do custo de geração. Essa métrica foi analisada separadamente porque alguns provedores cobram saída de forma diferente da entrada.
Latência	Indica viabilidade operacional. Uma arquitetura pode reduzir <i>tokens</i> e ainda assim ficar mais lenta se adicionar etapas sequenciais.
Chamadas de ferramentas	Mede o grau de exploração operacional do agente. Muitas chamadas podem indicar investigação necessária ou desperdício de roteamento.
D1	Controle de honestidade sobre cobertura indisponível. Foi usado para identificar arquiteturas que respondem demais quando deveriam reconhecer limitação dos dados.

Fonte: Elaboração própria.

O capítulo mantém *tokens* de entrada e *tokens* de saída separados porque eles representam componentes diferentes do custo. Além disso, a análise aplica uma estimativa monetária com preços de referência do OpenRouter para comparar a eficiência econômica das melhores configurações. Essa estimativa não substitui a cobrança real do provedor utilizado nas rodadas, mas evita tratar todos os *tokens* como se tivessem o mesmo preço. O mesmo cuidado vale para latência: ela não depende apenas da quantidade de *tokens*, mas também do modelo, da infraestrutura, do número de chamadas intermediárias e da serialização das ferramentas.

Também é importante separar três tipos de melhoria. A primeira é melhoria de qualidade, quando a arquitetura aumenta o número de OK ou melhora a nota média sem trocar o modelo. A segunda é melhoria de eficiência, quando preserva a qualidade e reduz *tokens*, ferramentas ou latência. A terceira é melhoria de perfil, quando o número total de acertos não muda, mas os testes acertados mudam. Esse último caso aparece

várias vezes nos resultados e é relevante, pois duas arquiteturas com 8/10 podem não ser equivalentes: uma pode acertar L9 e falhar L6, enquanto outra faz o contrário.

5.2 CUSTO TÉCNICO, PREÇO POR TOKEN E EFICIÊNCIA ECONÔMICA

A leitura inicial por *tokens* é necessária, mas não suficiente. Dois modelos podem consumir a mesma quantidade de *tokens* e gerar custos monetários muito diferentes, porque provedores cobram valores distintos para entrada e saída. Portanto, a eficiência econômica do *harness* deve ser medida por uma combinação de resultado, *tokens* consumidos e preço por milhão de *tokens*.

Para tornar essa comparação possível nesta versão preliminar, foi usada uma tabela de preços de referência do OpenRouter. Esses valores não representam uma fatura do Ollama nem necessariamente o preço real do provedor usado nas rodadas; eles funcionam como normalização externa para comparar as arquiteturas sob uma mesma base comercial. Como o *benchmark* não registrou *cache hit*, *cache write* ou *cache read* por chamada, a estimativa usa o preço de entrada sem *cache* e o preço de saída:

$$\hat{C}_m = \frac{T_{in}}{10^6} P_{in,m} + \frac{T_{out}}{10^6} P_{out,m}$$

Nessa fórmula, $\hat{C}_m$ representa o custo monetário estimado do modelo m , T_{in} e T_{out} são os *tokens* de entrada e saída medidos, e $P_{in,m}$ e $P_{out,m}$ são os preços por milhão de *tokens* de entrada e saída. Essa forma separa três dimensões que antes podiam ficar misturadas: acerto factual, eficiência técnica em *tokens* e custo econômico estimado.

Mesmo com essa limitação, a métrica de custo estimado por OK muda a interpretação dos resultados. Ela mostra quanto custaria, em uma referência comercial homogênea, produzir cada teste concluído corretamente. Quanto menor o valor de $\hat{C}_m/\text{OK}$, melhor a eficiência econômica da combinação modelo-arquitetura.

Essa tabela muda a leitura de alguns resultados. DeepSeek V4 Flash e MiniMax M2.7 chegaram ao mesmo número de acertos, mas o DeepSeek custou aproximadamente 3,2 vezes mais em dólar estimado e consumiu muito mais *tokens* por acerto. O Nemotron 3 Super, que antes era visto principalmente como um caso de alta latência, torna-se muito competitivo quando se considera preço por *token*: o custo estimado por OK foi o menor do recorte. O Gemma4 31B também melhora na leitura econômica, pois entrega 8/10 com custo estimado menor que o GPT-OSS 120B *baseline*. Já o Kimi K2.6 mostra o efeito oposto: seu consumo de *tokens* não é o pior, mas o preço unitário de entrada e saída torna o custo por acerto o mais alto entre as melhores configurações.

Quadro 9 – Custo técnico e custo monetário estimado nas melhores configurações

Modelo	Arquitetura	OK	Tokens/OK	US\$ est.	US\$/OK	Leitura de custo
Nemotron 3 Super	ACC+STR+CP	9	90.269	0,085	0,009	Melhor custo estimado por acerto, apesar de maior latência.
MiniMax M2.7	AT+TR+STR+CP	9	61.702	0,168	0,019	Melhor fronteira prática entre 9/10, baixo <i>token</i> e custo ainda competitivo.
DeepSeek V4 Flash	AT	9	649.717	0,536	0,060	Mesmo 9/10, mas custo estimado muito maior por alto consumo de entrada.
Gemma4 31B	AT+STR+CP	8	100.995	0,102	0,013	Melhor ganho balanceado: mais acertos, menos <i>tokens</i> e bom custo estimado.
GPT-OSS 120B	baseline	8	372.568	0,124	0,015	Forte como base e barato por <i>token</i> , mas sem ganho arquitetural relevante.
Kimi K2.6	AT+MB	8	173.344	1,149	0,144	Bom acerto, mas preço unitário alto transforma <i>tokens</i> moderados em custo elevado.
MiniMax M3	AT	8	301.650	0,818	0,102	Empata em acerto, mas saída cara e maior geração pesam no custo.
GPT-OSS 20B	RTS	6	468.356	0,089	0,015	Barato em dólares, mas baixo teto de acerto reduz atratividade como solução principal.

Fonte: Elaboração própria com base na planilha consolidada de *benchmark* e preços de referência do OpenRouter.

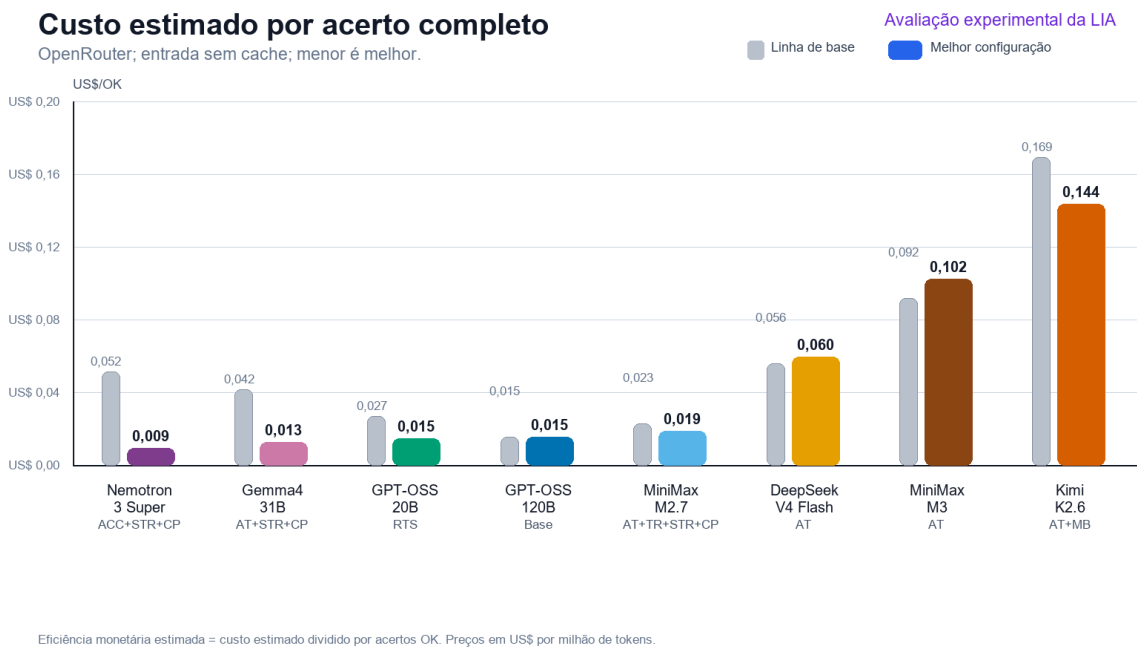


Figura 2 – Custo monetário estimado por acerto nas melhores configurações

Fonte: Elaboração própria com base na planilha consolidada de *benchmark* e preços de referência do OpenRouter.

A Figura 2 evidencia que eficiência real não é equivalente a menor total de *tokens*. GPT-OSS 20B, por exemplo, é barato em dólares estimados, mas acertou apenas 6/10 na melhor configuração; por isso, não supera alternativas com maior

acerto. MiniMax M2.7 permanece uma das melhores opções práticas por combinar 9/10 com custo estimado baixo. Nemotron 3 Super aparece como a melhor eficiência monetária por acerto, mas sua latência precisa ser considerada antes de uma escolha operacional. Portanto, a decisão final deve usar uma fronteira de múltiplas dimensões: OK, nota, D1, *tokens*, custo monetário e latência.

5.3 PORTE, PREÇO POR TOKEN E CONTROLE DE VIÉS ENTRE MODELOS

A matriz experimental foi executada com modelos de portes e perfis diferentes para evitar que a avaliação do *harness* ficasse enviesada por um único comportamento de modelo. Se todos os testes fossem feitos apenas com um modelo muito forte, uma arquitetura poderia parecer desnecessária porque o próprio modelo já resolveria parte do roteamento, da memória e da seleção de ferramentas. Se fossem feitos apenas com um modelo pequeno ou instável, a conclusão oposta também seria frágil: qualquer ganho poderia refletir apenas compensação de limitações daquele modelo específico, não uma melhoria geral do sistema.

Por isso, os modelos foram separados em faixas de capacidade esperada e capacidade observada. Essa separação não é uma escala absoluta de inteligência, mas um instrumento metodológico. O objetivo é observar se uma intervenção arquitetural melhora apenas modelos menores, se continua útil em modelos maiores, se reduz custo quando o acerto já é alto, ou se prejudica modelos que já executam bem o fluxo V2 da LIA. Em outras palavras, o *benchmark* procura separar três efeitos que podem se confundir: capacidade intrínseca do modelo, qualidade do *harness* e custo operacional da combinação modelo-arquitetura.

O tamanho em parâmetros também precisa ser tratado com cuidado. Nos modelos abertos ou explicitamente nomeados por porte, como GPT-OSS 20B, GPT-OSS 120B e Gemma4 31B, o valor foi registrado diretamente pelo identificador usado no experimento. Nos modelos servidos em nuvem com *aliases* proprietários, como MiniMax, Nemotron, DeepSeek e Kimi, a caracterização foi tratada como referência de contexto, não como variável experimental controlada. O objetivo da tabela não é provar superioridade por tamanho, mas mostrar por que modelos de portes, preços e perfis diferentes foram usados para reduzir viés na avaliação do *harness*.

Os preços do Quadro 10 seguem a referência do OpenRouter em dólares por milhão de *tokens*. Quando havia preços de *cache*, eles não foram usados na estimativa principal, porque o *benchmark* não mede *cache read/write* por chamada. Assim, os valores representam uma comparação conservadora com entrada sem *cache* e saída padrão.

A fonte de preços usada para a normalização econômica foi a API pública de modelos do OpenRouter, consultada em 17 de junho de 2026: <https://openrouter.ai/api/v1/models>.

Quadro 10 – Porte, categoria de capacidade e preço por milhão de *tokens* dos modelos avaliados

Modelo	Tamanho / contexto	Categoria usada	US\$/M <i>in/out</i>	Papel no <i>benchmark</i>
GPT-OSS 20B	21B total / 3,6B ativo; 128K	baixo / entrada	0,029 / 0,14	Limite inferior de capacidade; testa se o <i>harness</i> compensa um modelo menor.
Gemma4 31B	30,7B dense; 256K	médio	0,12 / 0,35	Modelo médio para medir ganho arquitetural em agente com capacidade moderada.
GPT-OSS 120B	117B total / 5,1B ativo; 128K	médio / alto	0,039 / 0,18	Modelo grande; testa se camadas adicionais ainda melhoram uma V2 já forte.
MiniMax M2.7	229B; 200K	médio / eficiente	0,25 / 1,00	Referência de fronteira entre alto acerto e baixo custo técnico por acerto.
MiniMax M3	aprox. 428B total / 23B ativo; 512K–1M	alto	0,30 / 1,20	Útil para observar empates de acerto com grande diferença de <i>tokens</i> .
Nemotron 3 Super	120B total / 12B ativo; 256K	médio	0,09 / 0,45	Caso em que ACC apresentou ganho relevante, com aumento de latência.
DeepSeek V4 Flash	284B total / 13B ativo; 1M	médio	0,09 / 0,18	Alto acerto, mas alto consumo; evidência <i>trade-off</i> entre qualidade e <i>tokens</i> .
Kimi K2.6	1,04T; 256K	alto	0,68 / 3,41	Modelo forte para observar troca de perfil entre L3, L6 e L9, mas com preço unitário alto.

Fonte: Elaboração própria com base nos identificadores dos modelos, planilha consolidada e preços de referência do OpenRouter.

O Quadro 10 também justifica a comparação entre modelos por categorias. O GPT-OSS 20B e o Gemma4 31B verificam se o *harness* consegue ampliar a capacidade prática de modelos menores ou médios. O GPT-OSS 120B verifica se o mesmo *harness* ainda é útil quando o modelo já possui capacidade maior. Os modelos proprietários em nuvem testam o comportamento em famílias treinadas de formas diferentes, com diferentes estilos de *tool calling*, resposta e uso interno de raciocínio. Essa variação reduz o risco de concluir que uma técnica é boa apenas porque foi ajustada aos erros de uma família específica.

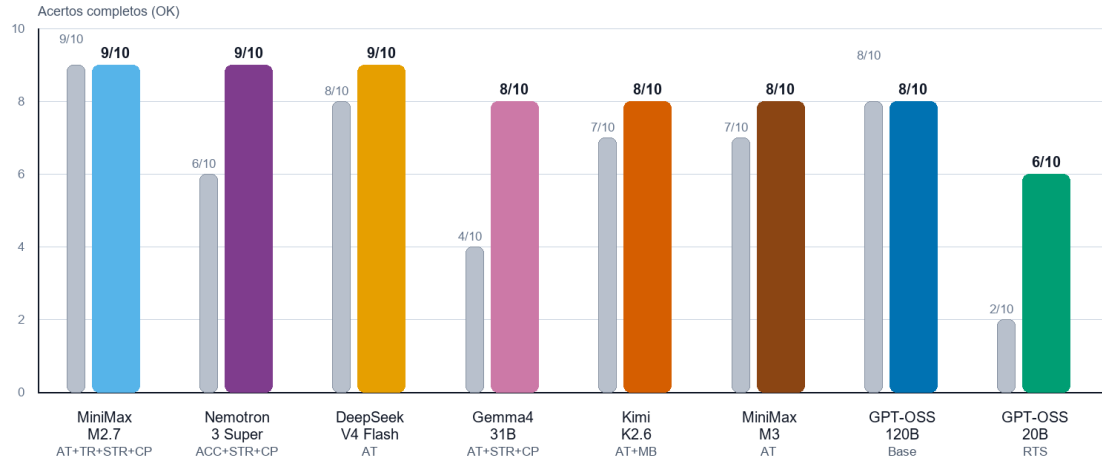
A Figura 3 apresenta a melhor configuração encontrada para cada modelo ou rodada consolidada. A visualização deixa claro que a análise não pode parar no número de OK. MiniMax M2.7, Nemotron 3 Super e DeepSeek V4 Flash chegaram a 9/10, mas por caminhos operacionalmente distintos. Gemma4 31B, Kimi K2.6, MiniMax M3 e GPT-OSS 120B chegaram a 8/10, porém com perfis de erro e custo muito diferentes. Assim, o empate em acertos não implica equivalência prática.

Melhor resultado por modelo

Valor = testes OK em L1-L10; a barra cinza fina mostra a linha de base.

Avaliação experimental da LIA

Linha de base Melhor configuração



A arquitetura destacada aparece abaixo de cada modelo. Quando a melhor configuração é a própria base, as duas barras coincidem.

Figura 3 – Melhor resultado por modelo nas configurações avaliadas

Fonte: Elaboração própria com base na planilha consolidada de *benchmark*.

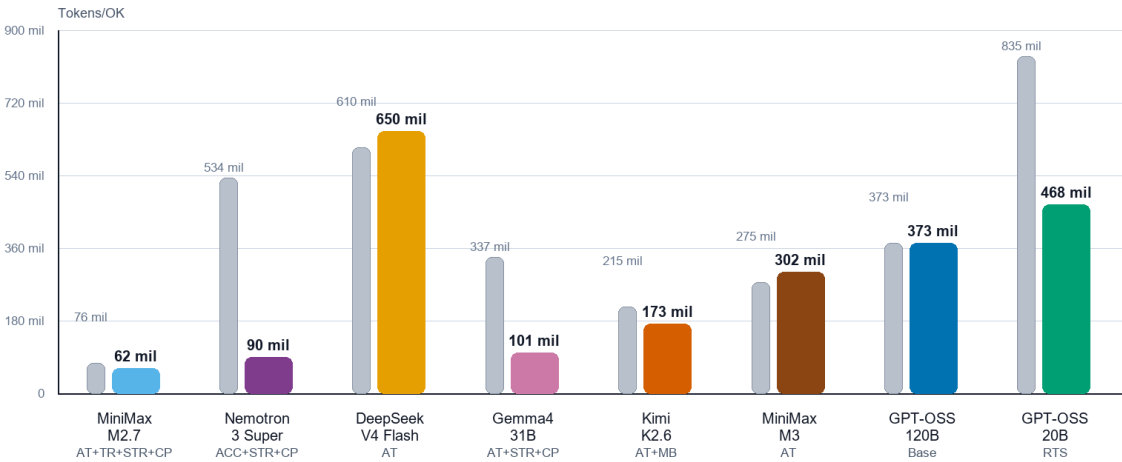
A Figura 4 mostra a dimensão de custo técnico antes da aplicação de preço. MiniMax M2.7 e Nemotron 3 Super atingiram 9/10 usando respectivamente cerca de 62 mil e 90 mil *tokens* por teste OK. DeepSeek V4 Flash também atingiu 9/10, mas consumiu cerca de 650 mil *tokens* por acerto. Essa leitura precisa ser combinada com a Figura 2: *tokens* explicam desperdício técnico, enquanto preço por *token* explica eficiência econômica.

Tokens por acerto completo

Total de tokens dividido por testes OK; menor é melhor.

Avaliação experimental da LIA

Linha de base Melhor configuração



Tokens/OK combina entrada e saída como custo técnico bruto. A barra cinza fina mostra a linha de base.

Figura 4 – *Tokens* por acerto nas melhores configurações de cada modelo

Fonte: Elaboração própria com base na planilha consolidada de *benchmark*.

As Figuras 5 e 6 resumem a lógica de fronteira usando custo monetário estimado por acerto completo. O eixo horizontal substitui o custo técnico em *tokens* por US\$/OK, calculado a partir dos *tokens* de entrada e saída e do preço por milhão de *tokens* de cada modelo. Os quadrados representam *baselines* puros, enquanto os círculos representam arquiteturas otimizadas selecionadas por fronteira, melhor acerto, melhor *score* ou menor custo por OK. Como não há uma arquitetura dominante em todos os modelos, a figura mantém mais de uma variante otimizada quando ela ocupa a fronteira ou esclarece o custo-benefício. O recorte monetário inclui apenas os modelos com preço explícito no Quadro 10.

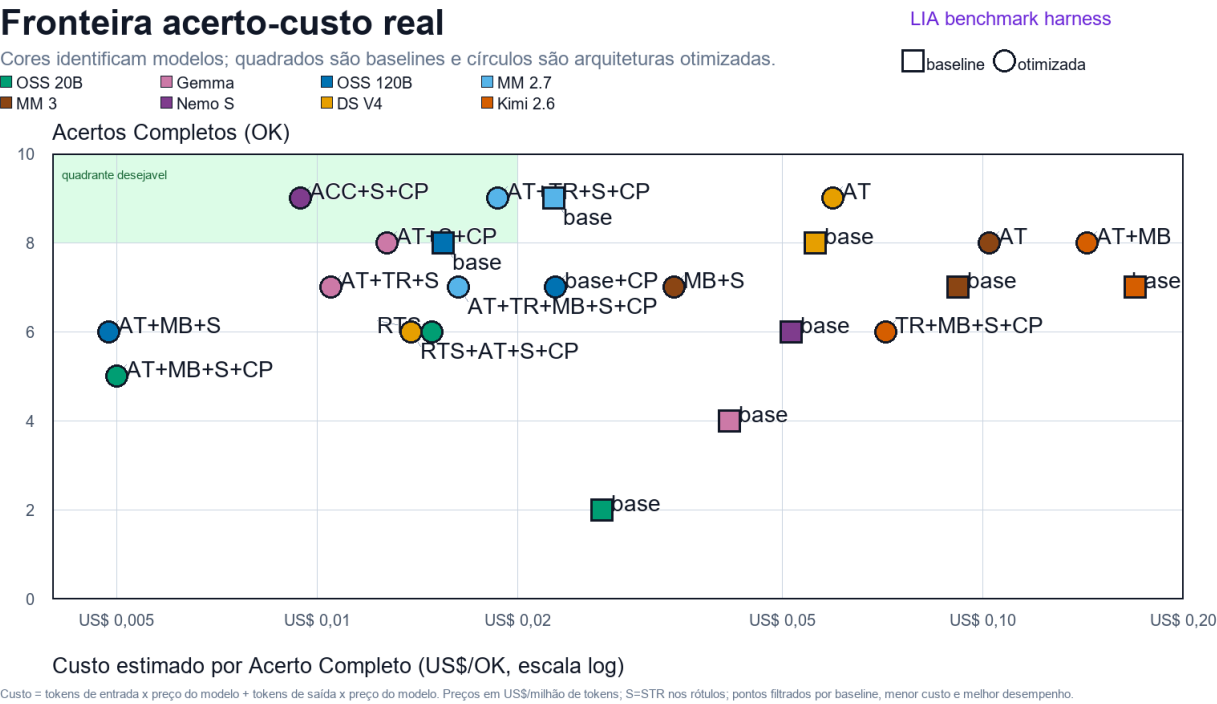


Figura 5 – Fronteira entre acertos completos e custo monetário estimado por acerto

Fonte: Elaboração própria com base na planilha consolidada de *benchmark*, nos preços por milhão de *tokens* e na seleção registrada em *data/cap5_frenteira_custo_pontos.csv*.

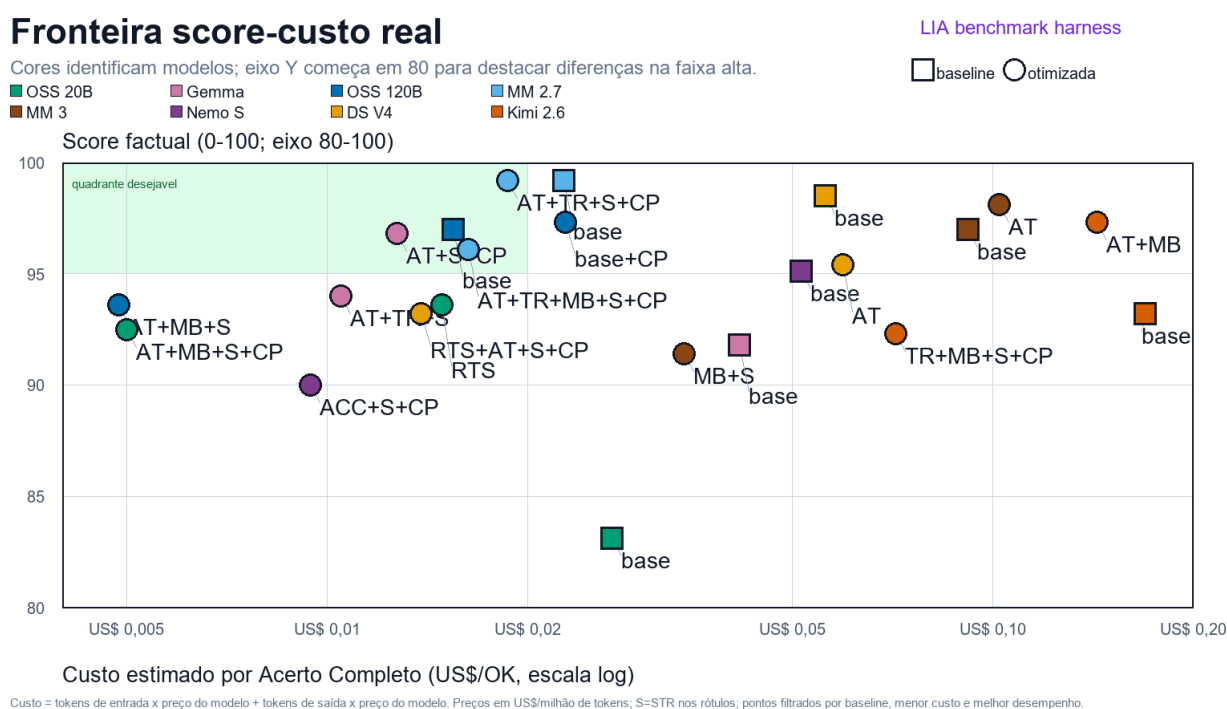


Figura 6 – Fronteira entre *score* factual e custo monetário estimado por acerto

Fonte: Elaboração própria com base na planilha consolidada de *benchmark*, nos preços por milhão de *tokens* e na seleção registrada em `data/cap5_fronreira_custo_pontos.csv`.

5.4 LEITURA GERAL: GANHO ARQUITETURAL NÃO UNIVERSAL

O comportamento agregado mostra que o *harness* conseguiu melhorar vários modelos, mas não por meio de uma única combinação dominante. Em Gemma4 31B, Nemotron 3 Super e GPT-OSS 20B, as melhores arquiteturas aumentaram consideravelmente o número de acertos em relação ao *baseline*. Em MiniMax M2.7 e GPT-OSS 120B, a linha de base já era forte, então as arquiteturas foram mais úteis para alterar custo, latência ou perfil de erro do que para aumentar acertos.

Esse resultado é coerente com o papel do *harness*. Ele não adiciona conhecimento novo ao modelo; ele organiza o caminho pelo qual o modelo acessa ferramentas, interpreta evidências e preserva restrições. Quando o modelo já executa bem esse caminho, camadas adicionais podem atrapalhar. Quando o modelo se perde em *prompts* longos, confunde ferramentas ou não preserva restrições, a arquitetura pode funcionar como uma forma de controle externo.

O quadro resume a primeira conclusão: o melhor *harness* depende da função objetivo. Se o objetivo for maximizar acertos, ACC+STR+CP no Nemotron 3 Super e AT+STR+CP no Gemma4 31B se destacam. Se o objetivo for reduzir *tokens* mantendo desempenho alto, MiniMax M2.7 com AT+TR+STR+CP, GPT-OSS 120B com TR+MB+STR e Nemotron 3 Super com MB+STR ou MB+STR+CP ficam mais interessantes. Se o ob-

Quadro 11 – Panorama inicial dos modelos e principais variantes

Modelo	Base	Base+CP	Destaque	Impacto contra base	Leitura principal
Gemma4 31B	4/10	5/10	AT+STR+CP	+4 OK; -40,1% <i>tokens</i> ; -28,7% latência	Melhor caso de ganho simultâneo em qualidade e eficiência.
Nemotron 3 Super	6/10	5/10	ACC+STR+CP	+3 OK; -74,7% <i>tokens</i> ; +64,1% latência	Maior ganho de acerto, mas com mais etapas e maior tempo total.
GPT-OSS 20B	2/10	4/10	RTS / STR	+4 OK; RTS +68,3% <i>tokens</i> ; STR +30,6% <i>tokens</i>	Arquitetura compensa limitação do modelo, mas o ganho custa mais.
MiniMax M2.7	9/10	9/10	AT+TR+STR+CP	0 OK; -18,4% <i>tokens</i> ; +45,4% latência	Modelo já forte; ganho principal é reduzir <i>tokens</i> e ferramentas.
GPT-OSS 120B	8/10	7/10	baseline	—	Modelo forte; camadas fixas não superaram a V2.
DeepSeek V4 Flash	8/10	6/10	AT	+1 OK; +19,8% <i>tokens</i> ; D1 falhou	Ganha L4, mas fica mais caro e menos seguro em cobertura.
Kimi K2.6	7/10	6/10	AT+MB	+1 OK; -7,8% <i>tokens</i> ; +6,8% latência	Melhora moderada e troca L3 por L6/L9.
MiniMax M3	7/10	7/10	AT e variantes eficientes	até +1 OK na rodada parcial; até -68,1% <i>tokens</i> na completa	Mesmo modelo mostrou dois efeitos: ganho textual com AT e economia com MB+STR.

Fonte: Elaboração própria com base na planilha consolidada de *benchmark*.

jetivo incluir segurança no D1, algumas variantes de alto desempenho precisam ser descartadas ou tratadas com cautela.

5.5 RESULTADOS POR ARQUITETURA

A análise por modelo mostra como cada família reagiu ao *harness*. A análise por arquitetura responde outra pergunta: quais combinações foram mais robustas quando observadas em vários modelos? Para evitar conclusões enganosas, a tabela abaixo considera apenas arquiteturas com cobertura em pelo menos cinco modelos. Isso é importante porque algumas arquiteturas pontuais obtêm resultado alto em poucos modelos, mas não possuem amostra suficiente para serem tratadas como solução geral.

O resultado mais importante dessa tabela é que “melhor arquitetura” depende do critério usado. Como o objetivo aqui é comparar intervenções arquiteturais, o *baseline* puro foi retirado da disputa. Ele continua como referência nas análises por modelo, mas não concorre como arquitetura otimizada. Se o critério for robustez média em maior número de modelos, *baseline*+CP foi a melhor intervenção. Isso não significa que ela foi sempre superior, mas que foi a configuração mais estável quando aplicada amplamente. Se o critério for acerto médio com planejamento explícito, AT fica próximo, mas com custo maior e regressões pontuais. Se o critério for eficiência por *token*, as arquiteturas com STR e combinações de CP dominam a parte superior: AT+TR+MB+STR, AT+STR+CP e AT+MB+STR entregaram mais acertos por milhão de *tokens* que as alterna-

Quadro 12 – Arquiteturas com maior cobertura e seus resultados médios

Arquitetura	Modelos	OK médio	Nota	Tokens médios	OK/Mtok	Conclusão
baseline+CP	9	6,33	93,2	2,85 M	2,23	Intervenção mais robusta no recorte filtrado, embora não seja a mais eficiente.
AT	8	5,88	92,4	2,69 M	2,19	Planejamento ajuda alguns testes, mas pode aumentar custo e variar por modelo.
STR	7	5,71	90,7	1,20 M	4,76	Boa eficiência; reduz ruído de ferramentas, mas perde cobertura em tarefas amplas.
AT+MB	7	5,71	92,4	2,50 M	2,28	Útil pontualmente quando memória e planejamento se alinham.
MB	7	5,71	93,4	2,93 M	1,95	Memória isolada preservou qualidade em alguns modelos, mas sem ganho de eficiência geral.
AT+STR+CP	8	5,62	91,3	0,86 M	6,51	Melhor compromisso entre eficiência e qualidade média no recorte filtrado.
AT+TR+STR	8	5,62	92,9	1,30 M	4,33	Alternativa intermediária, com ganho de custo e boa nota média.
AT+MB+STR	8	5,50	90,2	0,92 M	5,96	Reduz custo, mas não superou as melhores em acerto médio.
AT+TR+STR+CP	9	5,44	90,9	1,28 M	4,24	Boa cobertura experimental, mas com resultado médio abaixo das melhores.
AT+TR+MB+STR	8	5,25	90,1	0,77 M	6,86	Melhor eficiência média por token, porém com menor acerto médio.

Fonte: Elaboração própria com base na planilha consolidada de *benchmark*.

tivas conservadoras.

A conclusão prática é que a arquitetura mais robusta não foi a mesma arquitetura mais eficiente. *baseline+CP* é uma boa escolha geral quando se quer evitar grandes perdas. *AT+STR+CP* é mais interessante quando se busca otimização agressiva sem derrubar tanto a qualidade, especialmente em modelos como Gemma4 31B. *AT+TR+MB+STR* é uma arquitetura econômica, mas seu menor acerto médio indica que ela deve ser usada seletivamente. Já ACC e RTS não apareceram como soluções gerais, apesar de terem sido decisivos em casos específicos: ACC no Nemotron 3 Super e RTS/STR no GPT-OSS 20B.

5.6 MELHOR ARQUITETURA POR TESTE

O *benchmark* foi desenhado para separar tipos de capacidade. Por isso, a melhor arquitetura geral não deveria necessariamente ser a melhor em todos os testes. A tabela a seguir mostra a arquitetura com melhor taxa de acerto por cenário, considerando apenas arquiteturas com pelo menos cinco modelos executados naquele teste. A linha *baseline* foi retirada dessa disputa para que a tabela destaque intervenções

arquiteturais, não a própria V2 sem otimização. Em caso de empate, a leitura considera nota média e custo médio em *tokens*.

Quadro 13 – Arquiteturas com melhor desempenho por cenário de teste

Teste	Arquitetura destaque	Taxa OK	Tokens médios	Conclusão do cenário
L1	AT+TR+MB+STR	100,0%	29.156	Busca direta melhorou quando roteamento, memória e seleção de ferramentas convergiram para a entidade correta.
L2	AT+TR+MB+STR	100,0%	29.603	Resolução textual se beneficiou de memória/roteamento, com menor custo que MB isolado.
L3	STR	57,1%	39.651	Cruzamento saúde-empresa continuou sensível; sem o <i>baseline</i> puro, a melhor alternativa foi restringir ferramentas sem comprimir demais a evidência.
L4	AT+STR+CP	88,9%	67.880	Dossiê cadastral confirmou a hipótese de ganho com planejamento, seleção de ferramentas e <i>payload</i> compacto.
L5	AT+TR+STR	87,5%	61.030	Retenção curta de estado melhorou com recuperação e restrição de ferramentas.
L6	baseline+CP	46,2%	233.185	Cadeia profunda continuou difícil; compactação ajudou, mas não resolveu a propagação de entidades.
L7	MB+STR	100,0%	31.181	Repetição de padrão confirmou utilidade de memória e lista curta de ferramentas.
L8	AT	88,9%	345.230	Dossiê multi-domínio exigiu cobertura ampla; planejamento explícito foi a melhor intervenção, ainda com custo elevado.
L9	ACC	60,0%	1.584.447	Memória controlada ajudou conversa longa, mas com custo alto; AT+TR+STR ficou em 50,0%.
L10	AT	55,6%	110.842	Síntese estratégica se beneficiou de planejamento de ferramentas, mas permaneceu instável.

Fonte: Elaboração própria com base na planilha consolidada de *benchmark*.

A tabela confirma parcialmente as hipóteses teóricas. A busca direta e resolução textual, L1 e L2, melhoraram com arquiteturas que combinam roteamento, memória e restrição de ferramentas. O dossiê objetivo, L4, melhorou com AT, STR e CP, como esperado. A repetição de padrão, L7, melhorou com MB e STR, também de acordo com a proposta da memória tópica e da redução do espaço de ferramentas.

Por outro lado, L3 e L8 mostram o limite das otimizações. Quando o *baseline* puro é retirado da disputa, STR e AT aparecem como melhores alternativas, mas não por margens confortáveis. Em teoria, seleção de ferramentas e compactação deveriam ajudar tarefas multi-etapas, mas na prática essas tarefas dependem de uma cadeia factual sensível: preservar CNES, recuperar CNPJ, consultar perfil empresarial, interpretar natureza jurídica e, em alguns casos, integrar CRM, município ou equipamentos. Quando o *harness* reduz demais o contexto ou força uma rota incompleta, a resposta fica mais barata, mas perde evidência crítica.

Os registros de qualidade reforçam essa leitura. Nas falhas de L3, os critérios pendentes apareciam frequentemente associados a sócios, CNPJ completo e consolidação empresarial. Em L6, os critérios pendentes concentravam chamadas como *buscar_empresa*, *socios_empresa*, *crm_conta* e *crm Equipamentos*; ou seja, a falha

não era apenas textual, mas de encadeamento operacional. Em L8 e L9, apareceram pendências ligadas a ferramentas territoriais, como `buscar_municipio`, `populacao_municipio` e `indicadores_municipio`, além de preservação de camadas de CRM e governança. Em L10, os erros se concentraram em verificar CRM para as três unidades e produzir uma priorização explícita. Portanto, a diferença entre arquiteturas não está só no OK: ela aparece no tipo de etapa que cada configuração tende a esquecer.

5.7 ADERÊNCIA ENTRE HIPÓTESE ARQUITETURAL E RESULTADO PRÁTICO

A análise pareada por componente comparou, quando possível, o mesmo modelo e o mesmo teste em duas arquiteturas que diferiam por apenas um componente. Essa leitura não prova causalidade absoluta, mas ajuda a verificar se o efeito observado combina com a hipótese teórica de cada técnica.

Quadro 14 – Efeito pareado dos componentes arquiteturais

Comp.	Pares	Ganhos	Perdas	Saldo OK	Leitura prática
AT	940	144	145	-1	Quase neutro no agregado; ajudou L2, L9 e L10, mas prejudicou L3 e L7.
STR	480	79	96	-17	Reduziu <i>tokens</i> em média e ajudou L4/L7, mas prejudicou L8 e L10 quando removeu ferramentas úteis.
TR	840	107	124	-17	Teve ganhos pontuais em L9, mas não foi robusto como ativação fixa.
MB	850	118	140	-22	Ajudou L1, L7 e parte de L9; piorou L3, L4 e L5 quando a memória competiu com evidência nova.
CP	840	113	135	-22	Ajudou L4, L2 e L6; prejudicou L1, L7 e L9 quando a compactação removeu sinais necessários.
RTS	380	40	100	-60	Não confirmou ganho geral; foi útil no GPT-OSS 20B, mas caro e instável como padrão fixo.
ACC	310	24	103	-79	Forte no Nemotron 3 Super e em conversa longa, mas caro e negativo no agregado.

Fonte: Elaboração própria com base na planilha consolidada de *benchmark*.

O resultado pareado é uma das conclusões mais importantes do trabalho. Nenhum componente deve ser ativado de forma cega para todos os modelos e todos os testes. O AT confirmou sua utilidade em planejamento e síntese, mas também criou inércia de ferramenta em tarefas nas quais o *baseline* já seguia uma rota melhor. A STR confirmou sua função de reduzir espaço de ação e custo, mas falhou quando o teste exigia cobertura ampla. A CP ajudou quando havia redundância no *payload*, mas prejudicou quando o detalhe removido era justamente o dado necessário para validação factual.

O MB teve comportamento ambíguo. Teoricamente, deveria ajudar continuidade de tópico; na prática, ajudou repetição de padrão e algumas buscas, mas não foi a melhor solução para L5, que era o teste explícito de retenção curta de estado. Isso sugere que memória tópica não substitui um estado de execução bem controlado. O ACC tam-

bém confirma essa tensão: quando funcionou, como no Nemotron 3 Super, melhorou testes difíceis e reduziu *tokens* de entrada; quando não funcionou, introduziu latência, custo de consolidação e perda de aderência. O RTS, por fim, teve o comportamento mais pontual: sua hipótese é forte para contexto longo e ruidoso, mas o *benchmark* da LIA usa ferramentas estruturadas, não apenas documentos longos recuperados por RAG. Por isso, a etapa adicional de recitação nem sempre trouxe evidência melhor que o fluxo original.

5.8 ANÁLISE DETALHADA POR MODELO

Esta seção detalha os principais modelos avaliados. Diferentemente de um *ranking* simples, a comparação é feita dentro de cada modelo, usando o respectivo *baseline* como referência. Isso evita concluir que uma arquitetura é melhor apenas porque foi testada em um modelo mais forte. As tabelas seguem um formato compacto inspirado em relatórios de *benchmark*: cada linha mostra uma arquitetura relevante, o número de testes OK em L1–L10, a nota média, *tokens* de entrada e saída, latência, chamadas de ferramentas, D1 e quais testes falharam.

A coluna de falhas deve ser lida com o conjunto fixo L1–L10: todos os testes que não aparecem nessa coluna foram acertados pela arquitetura. Essa escolha deixa a comparação mais legível, especialmente nos modelos com 8/10 ou 9/10. O controle D1 aparece separado porque não mede uma pergunta respondível, mas a capacidade do agente de reconhecer ausência de cobertura suficiente. Portanto, uma arquitetura com muitos OK e D1 falho não é automaticamente a melhor opção operacional.

Além do resultado final, a análise considera a função exercida por cada otimização dentro do agente. AT tende a atuar como planejador de rota e seleção inicial de ferramentas; TR como mecanismo de redução do espaço de ferramentas disponíveis; MB como continuidade leve de tópico; STR como restrição estrutural da execução; CP como compactação de *payloads* e evidências; ACC como substituição do histórico por estado cognitivo; e RTS como separação entre extração de evidência e resposta final. Os resultados mostram que essas funções não são neutras: a mesma otimização pode reduzir custo em um modelo, induzir perda de cobertura em outro ou alterar quais testes são resolvidos.

5.8.1 Gemma4 31B: ganho simultâneo em acerto, *tokens* e latência

O Gemma4 31B é o caso mais claro de otimização arquitetural bem-sucedida. O *baseline* acertou apenas 4/10, mas não por incapacidade completa: ele acertou L3, L7, L8 e L9, ou seja, resolveu cruzamento de bases, repetição de padrão, dossiê integrado e conversa longa. A falha estava no aproveitamento operacional do agente em tarefas diretas e em retenção curta, como L1, L2, L4 e L5.

Quadro 15 – Destaques de arquiteturas para Gemma4 31B

Arquitetura	OK	Score	In M	Out k	Total k	Lat. min	Ferr.	D1	Falhas L1–L10
baseline	4	91,8	1,33	21,0	1348,7	15,7	81	OK	L1, L2, L4, L5, L6, L10
baseline+CP	5	95,5	1,49	20,7	1508,6	14,0	79	OK	L1, L3, L4, L6, L10
AT+STR	7	95,9	0,58	20,9	605,6	16,7	78	OK	L2, L4, L6
AT+TR+STR	7	94,0	0,55	20,9	572,4	12,1	86	OK	L4, L6, L9
AT+STR+CP	8	96,8	0,79	21,0	808,0	11,2	88	OK	L3, L6
ACC	7	94,0	2,77	45,6	2814,5	29,1	114	OK	L2, L5, L6

Fonte: Elaboração própria com base na planilha consolidada de *benchmark*. Verde indica melhor valor da coluna no recorte; linha cinza indica *baseline*.

A compactação isolada melhorou pouco: *baseline*+CP subiu para 5/10, ganhou L2 e L5, mas perdeu L3. Isso indica que CP reduziu ruído em buscas e navegação curta, mas também pode ter removido detalhes úteis para cruzamento entre bases. O ganho real apareceu com AT+STR+CP: 8/10, nota 96,8, redução de 40,1% no total de *tokens* e queda de latência de 942,0 s para 671,5 s. A arquitetura passou a acertar L1, L2, L4, L5, L7, L8, L9 e L10, falhando apenas L3 e L6.

As alternativas também são relevantes. AT+STR e AT+TR+STR chegaram a 7/10 e cortaram mais de metade dos *tokens*, mas perderam algum teste importante em relação à melhor configuração. ACC também atingiu 7/10, porém com 2,82 milhões de *tokens* totais e latência de 30,5 min, o que o torna menos atraente para esse modelo. Portanto, para o Gemma4 31B, AT+STR+CP é o melhor equilíbrio; AT+STR e AT+TR+STR são alternativas quando a prioridade é reduzir custo com pequena perda de qualidade.

No Gemma4 31B, a atuação das otimizações foi complementar. AT parece ter corrigido a desorientação inicial do *baseline* em tarefas diretas, pois as variantes com AT passaram a resolver L1 ou L10 com mais consistência. STR reduziu o espaço de ação e evitou exploração excessiva, explicando a queda forte de *tokens* em AT+STR e AT+TR+STR. CP, quando isolado, melhorou buscas e retenção curta, mas prejudicou L3; quando combinado com AT e STR, ele removeu ruído sem tirar sinais suficientes para L1, L2, L4 e L5. Já TR produziu a configuração mais econômica em *tokens*, mas a perda de L9 indica que a filtragem de ferramentas ou contexto ficou restritiva demais para conversa longa. O ACC atuou como memória estruturada, mas não trouxe ganho proporcional ao custo: ajudou em alguns testes, porém adicionou latência e saída demais para um modelo que respondeu melhor a planejamento e compactação simples.

5.8.2 Nemotron 3 Super: máximo acerto com ACC, eficiência com MB+STR

O Nemotron 3 Super já partiu de um *baseline* razoável, com 6/10, mas as falhas estavam nos testes que exigem organização de vários saltos: L3, L4, L6 e L10. O modelo era forte em busca, retenção curta, padrão, dossiê integrado e conversa longa, mas tinha dificuldade em encadear governança, CRM e síntese estratégica de forma

Quadro 16 – Destaques de arquiteturas para Nemotron 3 Super

Arquitetura	OK	Score	In M	Out k	Total k	Lat. min	Ferr.	D1	Falhas L1–L10
baseline	6	95,1	3,15	57,8	3205,8	15,0	50	OK	L3, L4, L6, L10
baseline+CP	5	93,9	3,65	64,4	3716,2	15,3	54	OK	L3, L4, L6, L7, L10
ACC+STR+CP	9	90,0	0,78	32,8	812,4	24,6	84	OK	L9
MB+STR	7	95,9	0,62	45,9	669,3	13,1	37	OK	L4, L5, L10
MB+STR+CP	7	96,6	0,75	50,3	803,5	13,8	52	OK	L1, L9, L10
STR+CP	7	94,4	0,80	49,7	853,4	10,9	50	OK	L3, L8, L10

Fonte: Elaboração própria com base na planilha consolidada de *benchmark*. Verde indica melhor valor da coluna no recorte; linha cinza indica *baseline*.

estável.

A variante ACC+STR+CP foi o maior salto de qualidade do conjunto: 9/10, com acertos adicionais justamente em L3, L4, L6 e L10. Ela falhou apenas L9. O resultado sustenta a hipótese do ACC: em tarefas longas ou com múltiplas dependências, substituir o *transcript* bruto por um estado cognitivo estruturado pode ajudar o modelo a manter objetivo, entidades e restrições ativas. Ao mesmo tempo, STR e CP reduziram o excesso de contexto. O total de *tokens* caiu 74,7% contra o *baseline*.

O custo pago foi latência e exploração: ACC+STR+CP levou 24,6 min e acionou 84 ferramentas, contra 15,0 min e 50 ferramentas no *baseline*. Por isso, MB+STR e MB+STR+CP são destaques operacionais mesmo com 7/10. Ambas reduzem aproximadamente três quartos dos *tokens*, mantêm D1 correto e entregam mais acertos que o *baseline*. Para esse modelo, há duas recomendações distintas: ACC+STR+CP quando qualidade máxima é o objetivo; MB+STR ou MB+STR+CP quando custo e tempo são mais importantes.

Neste modelo, o ACC foi a otimização com atuação mais visível. Ele compensou exatamente o tipo de falha do *baseline*: perda de organização em cadeias longas e sínteses finais. Ao manter um estado cognitivo explícito, a arquitetura conseguiu preservar objetivo, entidade central e restrições enquanto o agente atravessava L3, L4, L6 e L10. A contrapartida foi operacional: o compressor não substituiu o trabalho de investigação, ele o coordenou, e por isso as chamadas de ferramenta subiram. MB teve outro papel: não aumentou tanto o teto de qualidade, mas reduziu contexto mantendo continuidade suficiente para mais um acerto que o *baseline*. STR funcionou como freio de custo, enquanto CP só foi positivo quando usado como parte de uma rota controlada; em *baseline*+CP, a compactação sem mudança de controle piorou o total de acertos, indicando que resumir *payload* não basta quando o problema principal é ordem de raciocínio.

5.8.3 GPT-OSS 20B: *harness* como compensação de capacidade

O GPT-OSS 20B foi o modelo que mais dependeu do *harness*. O *baseline* acertou apenas L3 e L5 e ainda falhou D1, o que torna a configuração fraca tanto

Quadro 17 – Destaques de arquiteturas para GPT-OSS 20B

Arquitetura	OK	Score	In M	Out k	Total k	Lat. min	Ferr.	D1	Falhas L1–L10
baseline	2	83,1	1,62	45,9	1670,0	8,7	68	falha	L1, L2, L4, L6, L7, L8, L9, L10
baseline+CP	4	91,1	1,82	41,4	1866,2	8,4	66	OK	L3, L4, L6, L7, L8, L10
RTS	6	93,6	2,74	69,3	2810,1	13,4	78	OK	L3, L6, L8, L10
STR	6	90,0	2,11	68,9	2181,8	10,2	67	OK	L2, L6, L7, L10
MB+STR	5	93,5	1,80	57,1	1857,2	9,7	54	OK	L3, L6, L8, L9, L10
AT+MB+STR+CP	5	92,5	0,66	42,4	700,0	7,2	58	OK	L4, L6, L8, L9, L10

Fonte: Elaboração própria com base na planilha consolidada de *benchmark*. Verde indica melhor valor da coluna no recorte; linha cinza indica *baseline*.

em qualidade quanto em controle de cobertura. Esse resultado é útil para o trabalho porque mostra o limite inferior: sem apoio arquitetural, um modelo menor pode não seguir bem o protocolo de ferramentas e restrições.

baseline+CP já trouxe uma melhoria importante, subindo para 4/10 e corrigindo D1. Ele ganhou L1, L2 e L9, mas perdeu L3. A melhor qualidade veio com RTS e STR, ambas com 6/10. RTS pareceu ajudar ao separar recitação de evidência e resolução, reduzindo a carga cognitiva do modelo na etapa final. STR, por sua vez, alcançou o mesmo número de acertos com custo menor que RTS.

O *trade-off* é forte. RTS aumenta o total de *tokens* em 68,3% e a latência em 54,0% contra o *baseline*. STR aumenta *tokens* em 30,6%, mas ainda é mais barata que RTS. Já AT+MB+STR+CP é a configuração mais interessante do ponto de vista de otimização prática: sobe de 2/10 para 5/10, corrige D1 e reduz *tokens* em 58,1%. Assim, para o GPT-OSS 20B, a arquitetura de maior acerto não é a mais eficiente; a escolha depende de aceitar ou não pagar *tokens* extras para recuperar capacidade.

No GPT-OSS 20B, as arquiteturas atuaram como próteses de controle para um modelo com baixa autonomia operacional. CP corrigiu D1 e melhorou L1, L2 e L9 porque reduziu a carga de contexto que o modelo precisava interpretar antes de agir. RTS teve efeito coerente com sua proposta: ao separar evidência e resolução, deu ao modelo uma segunda etapa mais curta e menos ruidosa, o que elevou o *score* e recuperou L4. O custo subiu porque a técnica duplica parte da interação com o modelo. STR também chegou a 6/10, mas por outro caminho: restringiu a execução e evitou parte do desvio de ferramentas, embora tenha perdido testes diferentes de RTS. AT+MB+STR+CP mostra o efeito de uma arquitetura mais agressiva de economia: ela não maximiza acerto, mas combina planejamento, memória leve, restrição e compactação para cortar *tokens* e ainda corrigir D1. Nesse modelo, portanto, as otimizações não apenas refinam uma capacidade existente; elas definem se o agente consegue operar.

Quadro 18 – Destaques de arquiteturas para MiniMax M2.7

Arquitetura	OK	Score	In M	Out k	Total k	Lat. min	Ferr.	D1	Falhas L1–L10
baseline	9	99,2	0,64	45,5	680,9	21,1	103	falha	L9
baseline+CP	9	99,2	1,19	46,2	1236,6	24,7	95	OK	L9
AT+TR+STR+CP	9	99,2	0,52	38,9	555,3	30,7	66	falha	L10
ACC+STR	9	99,2	0,65	106,9	752,6	74,3	93	OK	L9
AT+MB	9	98,5	1,79	50,1	1842,3	37,4	107	falha	L9
STR	7	97,2	0,56	36,8	597,0	28,0	79	falha	L3, L9, L10

Fonte: Elaboração própria com base na planilha consolidada de *benchmark*. Verde indica melhor valor da coluna no recorte; linha cinza indica *baseline*.

5.8.4 MiniMax M2.7: teto alto, controle D1 e troca de perfil

O MiniMax M2.7 foi forte desde o *baseline*: 9/10, nota 99,2, falhando apenas L9. Por isso, o papel das arquiteturas não foi aumentar muito o número bruto de acertos, mas trocar perfil de erro, reduzir *tokens* e melhorar controle. O ponto fraco do *baseline* foi D1: o modelo respondeu além da cobertura disponível.

baseline+CP manteve 9/10 e corrigiu D1, mas quase dobrou o custo técnico, subindo de 680,9 mil para 1,24 milhão de *tokens*. AT+TR+STR+CP manteve 9/10 e reduziu *tokens* em 18,4%, além de diminuir ferramentas de 103 para 66. O detalhe decisivo é que ela acertou L9, mas passou a falhar L10 e também falhou D1. Portanto, não é simplesmente melhor: ela troca conversa longa por síntese estratégica e perde honestidade de cobertura.

ACC+STR também manteve 9/10 e corrigiu D1, mas gerou 106,9 mil *tokens* de saída e latência de 74,3 min. Esse é um exemplo importante para o *benchmark*: uma arquitetura pode parecer ótima em acerto e ainda ser operacionalmente ruim. Para o MiniMax M2.7, *baseline*+CP é a opção segura, AT+TR+STR+CP é a opção de eficiência com risco em D1, e ACC+STR é útil como evidência de controle, mas caro demais como padrão.

No MiniMax M2.7, as otimizações atuaram sobre um modelo que já sabia resolver quase tudo. Por isso, o efeito principal não foi criar capacidade, mas mudar o regime de controle. CP funcionou como camada de segurança: manteve 9/10 e corrigiu D1, mas aumentou muito o custo porque reestruturou *payloads* sem reduzir suficientemente a quantidade total de evidência processada. AT+TR+STR+CP atuou como rota enxuta: planejou, restringiu ferramentas e compactou o contexto, reduzindo *tokens* e chamadas, mas a combinação ficou restritiva ou assertiva demais para D1 e L10. ACC, por sua vez, preservou controle de cobertura e qualidade, mas deslocou custo para saída e tempo; o estado cognitivo ajudou o agente a não violar D1, porém estimulou respostas longas e consolidações caras. Esse caso mostra que, em modelos fortes, uma arquitetura pode ser julgada mais pelo que economiza ou pelo risco que introduz do que pelo número bruto de acertos.

5.8.5 GPT-OSS 120B: *baseline* forte e *harness* seletivo

Quadro 19 – Destaques de arquiteturas para GPT-OSS 120B

Arquitetura	OK	Score	In M	Out k	Total k	Lat. min	Ferr.	D1	Falhas L1–L10
<i>baseline</i>	8	97,0	2,93	53,5	2980,5	22,7	118	OK	L6, L10
<i>baseline</i> +CP	7	97,3	3,79	66,5	3853,2	22,1	110	OK	L1, L8, L10
TR+MB+STR	7	96,3	0,90	43,9	942,9	12,3	66	OK	L6, L8, L10
TR+STR+CP	7	95,5	1,93	56,8	1991,1	17,9	66	OK	L6, L8, L10
RTS+STR+CP	6	95,4	0,62	49,2	669,8	17,0	54	OK	L6, L8, L9, L10
AT+STR+CP	6	96,2	1,54	52,5	1597,2	16,3	73	OK	L6, L8, L9, L10

Fonte: Elaboração própria com base na planilha consolidada de *benchmark*. Verde indica melhor valor da coluna no recorte; linha cinza indica *baseline*.

O GPT-OSS 120B mostra o limite oposto ao GPT-OSS 20B. O *baseline* já obteve 8/10, nota 97,0 e D1 correto. Ele falhou L6 e L10, mas acertou os demais testes sem necessidade de camadas adicionais. Isso sugere que, para modelos maiores, a V2 da LIA já fornece um *harness* suficiente para boa parte do *benchmark*.

baseline+CP não melhorou o total: caiu para 7/10, embora tenha resolvido L6. A perda de L1 e L8 mostra que a compactação pode deslocar a competência do modelo. Ela ajudou uma cadeia profunda, mas prejudicou busca direta e dossiê multi-domínio. As variantes TR+MB+STR e TR+STR+CP também ficaram em 7/10, mas com grande redução de custo. TR+MB+STR cortou 68,4% dos *tokens* e 45,6% da latência.

A conclusão para esse modelo é que nenhuma arquitetura fixa superou o *baseline* em qualidade. Ainda assim, as variantes eficientes são úteis se o sistema aceitar perder um teste para economizar contexto e tempo. O GPT-OSS 120B, portanto, defende uma estratégia seletiva: usar *baseline* para tarefas críticas e rotas compactas como TR+MB+STR quando custo e latência pesarem mais que maximização de acerto.

No GPT-OSS 120B, o *baseline* já exercia internamente parte do que o *harness* tentava impor externamente: selecionar ferramentas, manter restrições e decidir quando aprofundar. Por isso, adicionar CP ou combinações restritivas nem sempre ajudou. CP deslocou a competência para L6, mas retirou estabilidade de L1 e L8, sugerindo perda de detalhe factual em tarefas que dependem de *payload* completo. TR+MB+STR foi a otimização mais interessante de custo porque cortou o espaço de ferramentas e preservou memória mínima, mas perdeu L6, L8 e L10. RTS com STR e CP foi ainda mais econômico, porém o passo de recitação não substituiu a necessidade de cruzamento amplo em testes multi-domínio. Assim, neste modelo, a atuação correta do *harness* é seletiva: usar otimizações como rotas baratas para perguntas simples, e não como substituto universal do *baseline*.

5.8.6 DeepSeek V4 Flash: maior acerto com regressão de custo e segurança

O DeepSeek V4 Flash começou forte, com *baseline* de 8/10, nota 98,5 e D1 correto. As falhas ocorreram em L4 e L9. A variante AT subiu para 9/10 ao corrigir L4,

Quadro 20 – Destaques de arquiteturas para DeepSeek V4 Flash

Arquitetura	OK	Score	In M	Out k	Total k	Lat. min	Ferr.	D1	Falhas L1–L10
baseline	8	98,5	4,78	97,1	4882,1	33,2	176	OK	L4, L9
baseline+CP	6	83,7	6,08	95,7	6172,0	33,3	197	OK	L2, L6, L7, L9
AT	9	95,4	5,74	106,4	5847,5	37,8	212	falha	L9
RTS+STR	7	95,2	0,98	62,3	1044,3	20,0	110	OK	L3, L8, L10
STR	7	92,8	1,56	98,0	1655,6	35,1	140	OK	L3, L6, L9
AT+TR+STR+CP	7	87,4	1,52	99,7	1620,9	35,0	128	falha	L3, L6, L10

Fonte: Elaboração própria com base na planilha consolidada de *benchmark*. Verde indica melhor valor da coluna no recorte; linha cinza indica *baseline*.

mantendo falha em L9. Em termos de acerto bruto, esse é um ganho real e coerente com a proposta do AT: planejar ferramentas e organizar a sequência ajuda no dossiê cadastral com governança e CRM.

O problema é que o ganho veio acompanhado de regressão operacional. AT aumentou *tokens* em 19,8%, elevou latência, aumentou chamadas de ferramentas e passou a falhar D1. Ou seja, a arquitetura melhorou uma pergunta respondível, mas piorou a capacidade de reconhecer cobertura insuficiente. Essa tensão é central para o *benchmark*, porque o objetivo do sistema não é responder sempre; é responder quando os dados permitem.

As variantes econômicas contam outra história. RTS+STR caiu para 7/10, mas reduziu *tokens* em 78,6% e corrigiu L9, perdendo L3, L8 e L10. STR também ficou em 7/10 com redução de 66,1% dos *tokens* e D1 correto. Assim, DeepSeek V4 Flash oferece três perfis claros: *baseline* como opção equilibrada e segura; AT como máximo acerto com risco; RTS+STR ou STR como rotas de baixo custo para cenários menos amplos.

No DeepSeek V4 Flash, AT atuou como acelerador de planejamento, mas também como fonte de inércia. Ele corrigiu L4 porque o dossiê cadastral exige decidir uma sequência de ferramentas e juntar governança, CRM e perfil institucional. Ao mesmo tempo, aumentou chamadas e falhou D1, sinal de que o planejamento tornou o agente mais propenso a continuar respondendo mesmo quando a cobertura era insuficiente. RTS atuou bem em L9 porque a conversa longa se beneficia de separar evidência relevante e resposta final; porém perdeu L3, L8 e L10, tarefas que exigem cruzamento simultâneo de domínios e não apenas evidência recitada. STR teve papel de contenção: reduziu custo e manteve D1, mas sacrificou cobertura. Portanto, nesse modelo, otimizações de planejamento aumentaram acerto, enquanto otimizações de restrição aumentaram eficiência e segurança, mas reduziram amplitude.

5.8.7 Kimi K2.6: ganho pontual em L6 e L9 com AT+MB

No Kimi K2.6, o *baseline* obteve 7/10, falhando L6, L9 e L10. A melhor variante foi AT+MB, com 8/10, nota 97,3 e redução de 7,8% no total de *tokens*. O ganho não

Quadro 21 – Destaques de arquiteturas para Kimi K2.6

Arquitetura	OK	Score	In M	Out k	Total k	Lat. min	Ferr.	D1	Falhas L1–L10
baseline	7	93,2	1,44	59,8	1503,4	16,5	83	OK	L6, L9, L10
baseline+CP	6	95,1	1,20	65,5	1263,8	18,6	91	OK	L3, L6, L9, L10
AT	7	94,0	1,34	68,5	1410,6	16,0	75	OK	L6, L9, L10
MB	7	96,6	1,41	64,1	1476,4	14,6	80	OK	L3, L6, L10
AT+MB	8	97,3	1,31	75,3	1386,7	17,6	91	OK	L3, L10
AT+MB+STR+CP	6	94,4	0,37	63,0	431,0	17,5	62	OK	L3, L8, L9, L10

Fonte: Elaboração própria com base na planilha consolidada de *benchmark*. Verde indica melhor valor da coluna no recorte; linha cinza indica *baseline*.

foi genérico: a variante passou a acertar L6 e L9, mas perdeu L3. Essa troca mostra que memória e planejamento ajudaram em cadeia profunda e conversa longa, mas prejudicaram o cruzamento saúde-empresa.

AT e MB isolados não aumentaram o total de acertos. AT manteve 7/10 com menor *token* total e menor latência; MB também manteve 7/10 e reduziu a latência. Já *baseline*+CP caiu para 6/10, apesar de nota média maior. Isso indica que o Kimi K2.6 não precisava principalmente de compactação: precisava de uma combinação de orientação de ferramenta e memória leve.

Assim, AT+MB é a melhor escolha para maximizar acerto nesse modelo, enquanto AT e MB são alternativas quando se deseja preservar o *baseline* com pequeno ganho operacional. O resultado também reforça que empates em OK escondem mudanças importantes: trocar L3 por L6/L9 pode ser aceitável ou não dependendo do produto.

No Kimi K2.6, a combinação AT+MB foi mais forte que os componentes isolados. AT sozinho pareceu organizar a rota sem alterar profundamente o perfil de acertos; MB sozinho ajudou continuidade e reduziu latência, mas também não elevou o teto. Juntos, planejamento e memória leve permitiram atravessar L6 e L9, que exigem continuidade de entidades e encadeamento mais longo. A perda de L3 sugere que a memória tópica ou a rota planejada pode ter fixado cedo demais uma entidade ou caminho de busca, reduzindo flexibilidade no cruzamento saúde-empresa. CP, por sua vez, não foi o principal fator de ganho: *baseline*+CP reduziu *tokens*, mas caiu em acerto, indicando que esse modelo precisava mais de continuidade e rota do que de *payload* menor.

5.8.8 MiniMax M3: eficiência na rodada completa e ganho textual na parcial

Na rodada completa do MiniMax M3, o *baseline* obteve 7/10 e falhou L3, L5 e L6. *baseline*+CP manteve 7/10, mas trocou o perfil: ganhou L6 e perdeu L9, com aumento de 56,8% nos *tokens*. A compactação, nesse caso, não foi uma melhoria geral; ela deslocou capacidade para a cadeia profunda e retirou desempenho de conversa longa.

As arquiteturas mais interessantes são as que mantiveram 7/10 com custo muito menor. MB+STR reduziu *tokens* em 68,1%, manteve D1 correto e quase não piorou latência. TR+MB+STR também manteve 7/10 e reduziu *tokens* em 59,8%, mas aumentou

Quadro 22 – Destaques de arquiteturas para MiniMax M3 na rodada completa

Arquitetura	OK	Score	In M	Out k	Total k	Lat. min	Ferr.	D1	Falhas L1–L10
baseline	7	97,0	1,86	70,9	1928,4	30,4	112	OK	L3, L5, L6
baseline+CP	7	89,0	2,95	70,7	3024,0	29,5	114	OK	L3, L5, L9
MB+STR	7	91,4	0,55	62,7	614,6	30,5	69	OK	L3, L6, L9
TR+MB+STR	7	96,6	0,72	58,1	775,4	42,5	102	OK	L3, L6, L9
AT+TR+STR	7	96,6	0,63	56,5	690,1	32,2	87	falha	L3, L4, L9

Fonte: Elaboração própria com base na planilha consolidada de *benchmark*. Verde indica melhor valor da coluna no recorte; linha cinza indica *baseline*.

latência. AT+TR+STR preservou 7/10 e cortou 64,2% dos *tokens*, porém falhou D1.

O MiniMax M3 completo mostra bem a diferença entre melhor qualidade e melhor eficiência. Nenhuma variante superou o *baseline* em acertos, mas várias mantiveram o mesmo patamar com custo muito menor. A configuração MB+STR é o destaque operacional porque combina economia alta, D1 correto e manutenção de 7/10.

No MiniMax M3 completo, MB+STR atuou como uma simplificação eficiente do agente. A memória leve preservou continuidade suficiente para não derrubar o total de acertos, enquanto a restrição estrutural reduziu ferramentas, *tokens* e desvios. TR+MB+STR adicionou seleção de ferramentas ao mesmo princípio e também economizou *tokens*, mas aumentou latência, sugerindo que o custo de decidir e reorganizar a rota não compensou totalmente. CP teve efeito de deslocamento: em *baseline*+CP, recuperou L6, mas perdeu L9 e aumentou *tokens*; isso indica que a compactação ajudou cadeia profunda, mas prejudicou conversa longa. AT+TR+STR manteve 7/10 com baixo custo, porém falhou D1, reforçando que planejamento mais restritivo pode aumentar confiança indevida quando o agente deveria reconhecer falta de cobertura.

A rodada parcial do MiniMax M3 é mantida no mesmo subcapítulo porque avalia o mesmo modelo em outro recorte de execução. Ela não deve ser lida como um segundo modelo, mas como evidência complementar sobre o comportamento do MiniMax M3 quando apenas *baseline*+CP e AT estavam disponíveis no consolidado.

Quadro 23 – Destaques de arquiteturas para MiniMax M3 na rodada parcial

Arquitetura	OK	Score	In M	Out k	Total k	Lat. min	Ferr.	D1	Falhas L1–L10
baseline+CP	8	94,3	2,90	72,6	2973,5	32,3	157	OK	L5, L9
AT	8	98,1	2,31	104,9	2413,2	37,0	127	OK	L5, L9

Fonte: Elaboração própria com base na planilha consolidada de *benchmark*. Verde indica melhor valor da coluna no recorte; linha cinza indica *baseline*.

Nessa rodada parcial, *baseline*+CP e AT obtiveram 8/10 e falharam os mesmos testes, L5 e L9. O ganho não está no acerto bruto, mas no perfil operacional.

AT reduziu *tokens* de entrada e chamadas de ferramentas, mas aumentou *tokens* de saída e latência. Isso sugere que o planejamento ajudou a organizar a resposta, porém induziu respostas mais longas. A conclusão é limitada pela ausência de *baseline*

puro nessa rodada, mas reforça a ideia de que AT pode melhorar organização textual sem necessariamente reduzir custo total.

Na rodada parcial, a atuação de AT foi mais textual do que decisória. Como *baseline*+CP e AT erraram os mesmos testes, o planejamento não mudou a cobertura resolvida. O que mudou foi a forma de execução: menos entrada e menos ferramentas, mas mais saída e mais tempo. Isso sugere que o modelo aceitou melhor a instrução de planejar e justificar a resposta, produzindo textos mais longos sem resolver L5 ou L9. Para o *harness*, esse é um alerta: otimizações que parecem reduzir custo no lado de entrada podem deslocar o custo para geração, especialmente em modelos que respondem de forma mais elaborada quando recebem estrutura explícita.

Síntese da análise por modelo

A primeira leitura dos resultados sustenta três conclusões. A primeira é que o *harness* pode melhorar significativamente um modelo, mas o efeito depende do modelo. Gemma4 31B e Nemotron 3 Super foram os exemplos mais fortes: ambos saíram de desempenhos medianos para configurações competitivas, e o ganho não veio apenas de trocar o modelo. GPT-OSS 20B também melhorou, mas o aumento de qualidade exigiu mais custo em algumas variantes.

A segunda conclusão é que acerto e eficiência precisam ser avaliados juntos. MiniMax M2.7, GPT-OSS 120B e MiniMax M3 mostram que uma arquitetura pode não aumentar *OK*, mas ainda ser relevante se reduzir *tokens*, ferramentas ou latência. Em sentido oposto, DeepSeek V4 Flash mostra que uma arquitetura pode aumentar acerto e ainda assim ser menos desejável se aumentar custo e falhar em D1.

A terceira conclusão é que o erro por teste importa tanto quanto o total de acertos. Várias arquiteturas mantiveram o mesmo número de *OK*, mas trocaram os testes que acertavam. Isso ocorreu em MiniMax M2.7, MiniMax M3 e GPT-OSS 120B. Portanto, a escolha arquitetural deve ser orientada por rota de uso: busca direta, dossiê multi-domínio, conversa longa, cadeia profunda ou síntese estratégica podem exigir *harnesses* diferentes.

Com a leitura por modelo, arquitetura e teste, a conclusão parcial fica mais precisa: o *harness* não deve ser tratado como uma configuração única, mas como um conjunto de rotas possíveis. Para perguntas simples e repetitivas, arquiteturas mais econômicas com STR e MB podem ser suficientes. Para dossiês objetivos, AT, STR e CP se mostraram fortes. Para cadeia profunda e cobertura parcial, a escolha precisa ser mais conservadora, pois compressão e filtragem podem remover evidências necessárias. Quando o custo monetário estimado é incluído, a decisão fica ainda mais dependente da fronteira entre acerto, preço por *token*, latência e segurança em D1.

Os dados usados nas tabelas e figuras foram preservados nos arquivos auxiliares do projeto, incluindo as métricas por modelo-arquitetura, o resumo por teste, o

efeito pareado dos componentes e a estimativa de custo com preços de referência. Essa rastreabilidade permite regenerar as figuras sempre que houver nova rodada de *benchmark* ou alteração na tabela de preços, sem transformar o corpo do texto em uma listagem de arquivos internos.

6 CONCLUSÃO

Este capítulo fecha a análise retomando o problema inicial, os objetivos definidos na introdução e os resultados obtidos com o *benchmark* da LIA. A conclusão não trata apenas do melhor modelo ou da melhor arquitetura isolada, mas do que os experimentos mostram sobre custo, latência, uso de ferramentas, controle de contexto e confiabilidade em um agente aplicado a bases de saúde, empresas e CRM.

6.1 SÍNTESE DO TRABALHO

Este trabalho tratou do problema de otimização de agentes de IA generativa com uso de ferramentas externas em um cenário de integração de bases de saúde, dados empresariais e informações de CRM. O problema central não era apenas responder a uma pergunta isolada, mas orquestrar corretamente consultas multi-etapas, preservar entidades, controlar contexto, evitar alucinações e declarar limitações quando os dados locais não sustentassem uma resposta.

Para isso, foi analisado e adaptado o *pipeline* multiagente da LIA, composto por carregamento de memória, roteamento, resolução de entidades, execução de ferramentas e composição final da resposta. Sobre esse *pipeline* foram implementadas e avaliadas diferentes estratégias arquiteturais, incluindo seleção semântica de ferramentas, compactação de *payload*, inferência de sequências de ferramentas, recuperação orientada por grafo, memória tópica, compressão cognitiva e separação entre recitação de evidência e resolução.

Também foi desenvolvida uma metodologia de avaliação por ablação. A suíte de testes foi ajustada para o ambiente local parcial, com cenários de complexidade crescente entre L1 e L10 e um controle diagnóstico D1 para verificar honestidade diante de cobertura territorial ausente. Essa metodologia permitiu medir acerto factual, latência, *tokens* de entrada e saída, ferramentas acionadas e falhas de infraestrutura.

6.2 ATENDIMENTO AOS OBJETIVOS

O objetivo geral do trabalho foi desenvolver e avaliar otimizações arquiteturais para reduzir custo, latência e instabilidade em agentes com ferramentas, preservando a qualidade factual em cenários de suporte a vendas complexas. Esse objetivo foi atingido de forma substancial, embora não absoluta. As arquiteturas implementadas demonstraram ganhos relevantes em alguns modelos e cenários, mas os resultados também mostraram que não há uma estratégia única superior para todos os casos.

Os objetivos específicos também foram atendidos. A arquitetura existente da LIA foi analisada e seu gargalo principal foi caracterizado como combinação de inflação de contexto, seleção ampla de ferramentas, repetição de decisões e dificuldade de preser-

var entidades em cadeias longas. O ambiente experimental local foi estruturado sem depender do sistema completo de produção, preservando o núcleo de orquestração necessário para o estudo.

A suíte de *benchmark* foi desenvolvida e refinada para medir capacidades viáveis no recorte disponível. Esse ponto foi especialmente importante, pois testes impossíveis ou excessivamente fáceis tornariam a comparação sem valor. As execuções finais diferenciaram recuperação simples, resolução textual, encadeamento multi-domínio, retenção de contexto, conversa longa, priorização estratégica e honestidade diante de cobertura parcial.

As estratégias arquiteturais propostas também foram implementadas e comparadas. A análise mostrou que STR, CP, AT, MB, ACC e RTS podem alterar significativamente o comportamento do agente, mas seus efeitos dependem do modelo, do tipo de tarefa e do custo interno de cada mecanismo. O resultado mais importante não foi apenas implementar as técnicas, mas identificar em quais situações elas melhoram, deslocam ou prejudicam o desempenho.

6.3 PRINCIPAIS RESULTADOS

Os resultados indicaram que a arquitetura do agente influencia de forma relevante a qualidade final, mas não substitui a escolha do modelo. No recorte principal discutido no Capítulo 5, as melhores combinações alcançaram até 9 acertos em 10 testes comparativos, mas nenhuma arquitetura foi universalmente superior. A principal conclusão é que o *harness* deve ser tratado como um conjunto de rotas de execução, não como uma configuração única aplicável a qualquer modelo.

O *minimax-m2.7* foi uma das melhores fronteiras técnico-operacionais: a variante AT+TR+STR+CP manteve 9/10, reduziu aproximadamente 18,4% dos *tokens* em relação ao *baseline* e preservou nota média de 99,2, embora com maior latência. Quando o preço por *token* é aplicado, o *nemotron-3-super* com ACC+STR+CP passa a ter o menor custo monetário estimado por acerto, mas também exige cautela por sua latência. O *deepseek-v4-flash* também atingiu 9/10 com AT, mas consumiu muito mais *tokens*, teve custo estimado maior e falhou no controle D1. Assim, resultados iguais em OK não representam a mesma qualidade operacional.

O *gemma4:31b* apresentou o ganho arquitetural mais claro: sua linha de base acertou 4/10 testes, enquanto AT+STR+CP alcançou 8/10, reduziu cerca de 40,1% dos *tokens* totais, diminuiu a latência e manteve custo monetário estimado competitivo. O *nemotron-3-super* também melhorou de forma expressiva, saindo de 6/10 para 9/10 com ACC+STR+CP; nesse caso, porém, o ganho de acerto veio acompanhado de maior latência. Esses dois casos mostram que as camadas arquiteturais podem ampliar muito a capacidade prática de modelos que, no *baseline*, não exploravam bem o fluxo de ferramentas.

Em modelos já fortes, o efeito foi diferente. O `gpt-oss:120b` manteve bom desempenho na V2 pura e as variantes testadas foram mais úteis para estudar custo e perfil de erro do que para superar o total de acertos. No `minimax-m3`, a rodada completa mostrou combinações que mantiveram o *baseline* com reduções expressivas de *tokens*, enquanto a rodada parcial mostrou AT alcançando 8/10, mas com aumento de saída e latência. Esse caso evidencia que a melhor arquitetura nem sempre é a que apenas acerta mais; em alguns cenários, a contribuição real é preservar qualidade com menor custo ou deslocar o perfil de erro.

A leitura por arquitetura reforçou esse ponto. `baseline+CP` foi a intervenção mais robusta em média, mas não a mais eficiente. `AT+STR+CP` apareceu como um dos melhores compromissos entre qualidade e economia, especialmente em modelos intermediários. Já `AT+TR+MB+STR` obteve alta eficiência por milhão de *tokens*, mas com menor acerto médio. A aplicação de preços por milhão de *tokens* acrescentou outra camada: modelos com consumo moderado podem ficar caros quando o preço unitário de saída é alto, enquanto modelos com mais *tokens* podem continuar competitivos se o preço por *token* for baixo. Portanto, a escolha final depende da função objetivo: maximizar acerto, reduzir custo monetário, preservar D1, minimizar latência ou priorizar certos tipos de tarefa.

No nível dos testes, o *benchmark* demonstrou capacidade discriminativa. L3 e L8 evidenciaram o risco de remover contexto factual em tarefas multi-etapas; L6 continuou difícil por exigir propagação correta de entidades em cadeia profunda; L9 favoreceu memória controlada, mas com custo elevado; e L10 permaneceu instável por combinar síntese estratégica, verificação de CRM e priorização explícita. O controle D1 também se mostrou necessário: algumas arquiteturas de alto desempenho em OK falharam justamente por responderem além da cobertura disponível, o que confirma que honestidade sobre ausência de dados deve ser medida como critério próprio.

6.4 LIMITAÇÕES E PRÓXIMOS PASSOS

A principal limitação do trabalho está no recorte local e parcial da plataforma. O ambiente utilizado permitiu controle experimental, repetibilidade e redução de custo, mas não representa toda a infraestrutura corporativa da LIA. O índice local de estabelecimentos, por exemplo, não possuía cobertura adequada para campos territoriais como UF e município. Por isso, parte dos testes precisou ser redesenhada para medir capacidades disponíveis no ambiente de avaliação, em vez de exigir respostas que dependeriam de bases ausentes.

A segunda limitação é operacional. Durante as rodadas, ocorreram *timeouts*, interrupções de conexão e limites de cota dos modelos em nuvem. O *benchmark* foi adaptado para salvar *checkpoints* e separar falhas de infraestrutura de falhas funcionais do agente, mas essas ocorrências ainda reduzem a simetria entre algumas

comparações. Em uma continuação do trabalho, seria importante repetir as melhores configurações em janelas menores e mais controladas, reduzindo o risco de comparar rodadas afetadas por condições externas diferentes.

Também houve limitações de instrumentação. Embora os *tokens* de entrada e saída tenham sido registrados, algumas arquiteturas com chamadas auxiliares, como MB, ACC e RTS, exigem contabilização separada de cada etapa interna. Sem essa medição completa, uma arquitetura pode aparentar ser mais econômica apenas por deslocar *tokens* para uma chamada que não aparece com clareza no custo principal. Esse ponto é especialmente relevante porque o Capítulo 5 mostrou que *tokens* de saída e preço por milhão de *tokens* alteram bastante a leitura de eficiência.

Outro ponto a evoluir é a pontuação. A nota factual já ajudou a diferenciar respostas corretas, degradadas e incorretas, mas ainda pode ser refinada com pesos explícitos para ferramentas obrigatórias, preservação de entidade, completude da resposta e declaração de indisponibilidade de dados. Isso reduziria casos em que uma resposta acerta itens secundários, mas falha em uma etapa essencial do raciocínio.

Os próximos passos técnicos mais importantes são: ampliar a cobertura das bases locais, registrar o custo interno de cada chamada auxiliar, repetir as melhores combinações em rodadas menores e testar perguntas derivadas de fluxos comerciais reais. Esses passos não mudam o objetivo do *benchmark*; eles tornam a comparação mais próxima do uso cotidiano da plataforma.

6.5 IMPACTOS DO TRABALHO

O principal impacto técnico do trabalho foi transformar a avaliação de agentes da LIA em um processo mensurável. Antes da estruturação do *benchmark*, uma arquitetura poderia parecer melhor apenas por produzir respostas mais fluentes ou por falhar silenciosamente em etapas internas. Com os relatórios de latência, *tokens*, ferramentas acionadas, status funcional e *traces*, tornou-se possível comparar propostas de forma mais objetiva.

Para a equipe e para a empresa, o resultado ajuda a tomar decisões menos baseadas em impressão e mais baseadas em evidência. Os dados mostram que escolher o modelo mais potente não é necessariamente a melhor alternativa operacional. Em algumas combinações, modelos menores ou intermediários alcançaram melhor compromisso entre acerto, custo e latência.

Para o produto, o trabalho reforça a importância de agentes que saibam declarar limites. Em cenários de suporte a vendas complexas, respostas inventadas sobre cobertura territorial, governança ou CRM podem induzir decisões comerciais equivocadas. O controle de honestidade e a separação entre falha factual e falha de infraestrutura são contribuições relevantes para aumentar a confiabilidade do sistema.

FECHAMENTO

O resultado central do trabalho é que otimizar agentes com ferramentas exige mais do que trocar o modelo de linguagem. A qualidade final depende da forma como o sistema seleciona ferramentas, organiza contexto, preserva entidades, controla memória, mede custo e reconhece os limites dos dados disponíveis. O *benchmark* desenvolvido tornou essas decisões comparáveis dentro da LIA, permitindo observar quando uma arquitetura aumenta acertos, quando apenas reduz custo e quando muda o perfil de erro.

A versão avaliada ainda não resolve todos os cenários, mas deixa um caminho técnico claro. Para perguntas simples e repetitivas, arquiteturas mais enxutas podem ser suficientes. Para dossiês e cadeias multi-etapas, o ganho depende de equilibrar planejamento, compactação, memória e cobertura de ferramentas. Para uso operacional, a escolha final não deve ser apenas o maior número de OK, mas a melhor fronteira entre acerto, custo monetário, *tokens*, latência e segurança diante de dados ausentes.

REFERÊNCIAS

- BOUSETOUANE, Fouad. AI Agents Need Memory Control Over More Context, 2026. Disponível em: <https://arxiv.org/abs/2601.11653>. Acesso em: 9 jun. 2026.
- BROWN, Tom B.; MANN, Benjamin; RYDER, Nick; SUBBIAH, Melanie; KAPLAN, Jared et al. Language Models are Few-Shot Learners. In: ADVANCES in Neural Information Processing Systems. [S.l.: s.n.], 2020. p. 1877–1901. Disponível em: <https://arxiv.org/abs/2005.14165>. Acesso em: 7 jun. 2026.
- DU, Yufeng et al. Context Length Alone Hurts LLM Performance Despite Perfect Retrieval, 2025. Disponível em: <https://arxiv.org/abs/2510.05381>. Acesso em: 9 jun. 2026.
- HU, Yuanzhe; WANG, Yu; MCAULEY, Julian. Evaluating Memory in LLM Agents via Incremental Multi-Turn Interactions. In: INTERNATIONAL Conference on Learning Representations. [S.l.: s.n.], 2026. Disponível em: <https://arxiv.org/abs/2507.05257>. Acesso em: 9 jun. 2026.
- HU, Yuyang; LIU, Shichun; YUE, Yanwei; ZHANG, Guibin; LIU, Boyang; ZHU, Fangyi; LIN, Jiahang et al. Memory in the Age of AI Agents: A Survey Forms, Functions and Dynamics, 2026. Disponível em: <https://arxiv.org/abs/2512.13564>. Acesso em: 9 jun. 2026.
- JIA, Jingyi; LI, Qinbin. AutoTool: Efficient Tool Selection for Large Language Model Agents, 2026. Disponível em: <https://github.com/jiajingyyyyyy/AutoTool>. Acesso em: 9 jun. 2026.
- JIANG, Huiqiang; WU, Qianhui; LIN, Chin-Yew; YANG, Yuqing; QIU, Lili. LLMLingua: Compressing Prompts for Accelerated Inference of Large Language Models. In: PROCEEDINGS of the 2023 Conference on Empirical Methods in Natural Language Processing. [S.l.: s.n.], 2023. p. 13358–13376. DOI: 10.18653/v1/2023.emnlp-main.825. Disponível em: <https://aclanthology.org/2023.emnlp-main.825/>. Acesso em: 9 jun. 2026.
- JIANG, Huiqiang; WU, Qianhui; LUO, Xufang; LI, Dongsheng; LIN, Chin-Yew; YANG, Yuqing; QIU, Lili. LongLLMLingua: Accelerating and Enhancing LLMs in Long Context Scenarios via Prompt Compression. In: PROCEEDINGS of the 62nd Annual Meeting of the Association for Computational Linguistics. [S.l.: s.n.], 2024. p. 1658–1677. DOI: 10.18653/v1/2024.acl-long.91. Disponível em: <https://aclanthology.org/2024.acl-long.91/>. Acesso em: 9 jun. 2026.
- LEWIS, Patrick et al. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In: ADVANCES in Neural Information Processing Systems. [S.l.: s.n.], 2020. Disponível em: <https://arxiv.org/abs/2005.11401>. Acesso em: 7 jun. 2026.

LIAO, Huanxuan; HU, Wen; XU, Yao; HE, Shizhu; ZHAO, Jun; LIU, Kang. Beyond Hard and Soft: Hybrid Context Compression for Balancing Local and Global Information Retention, 2025. Disponível em: <https://arxiv.org/abs/2505.15774>. Acesso em: 9 jun. 2026.

LIU, Nelson F.; LIN, Kevin; HEWITT, John; PARANJAPE, Ashwin; BEVILACQUA, Michele; PETRONI, Fabio; LIANG, Percy. Lost in the Middle: How Language Models Use Long Contexts. **Transactions of the Association for Computational Linguistics**, v. 12, p. 157–173, 2024. DOI: 10.1162/tacl_a_00638. Disponível em: <https://aclanthology.org/2024.tacl-1.9/>. Acesso em: 7 jun. 2026.

MIALON, Grégoire et al. Augmented Language Models: a Survey. **Transactions on Machine Learning Research**, 2023. Disponível em: <https://openreview.net/forum?id=jh7wH2AzKK>. Acesso em: 7 jun. 2026.

MODEL CONTEXT PROTOCOL. **Model Context Protocol Specification: Overview**. 2025. Disponível em: <https://modelcontextprotocol.io/specification/2025-06-18/basic/index>. Acesso em: 7 jun. 2026.

QU, Changle; DAI, Sunhao; WEI, Xiaochi; CAI, Hengyi; WANG, Shuaiqiang; YIN, Dawei; XU, Jun; WEN, Ji-Rong. Towards Completeness-Oriented Tool Retrieval for Large Language Models. In: PROCEEDINGS of the 33rd ACM International Conference on Information and Knowledge Management. [S.l.: s.n.], 2024. p. 1–11. DOI: 10.1145/3627673.3679847. Disponível em: <https://doi.org/10.1145/3627673.3679847>. Acesso em: 9 jun. 2026.

SCHICK, Timo; DWIVEDI-YU, Jane; DESSÌ, Roberto; RAILEANU, Roberta; LOMELI, Maria; HAMBRO, Eric; ZETTLEMOYER, Luke; CANCEDDA, Nicola; SCIALOM, Thomas. Toolformer: Language Models Can Teach Themselves to Use Tools. In: ADVANCES in Neural Information Processing Systems. [S.l.: s.n.], 2023. Disponível em: <https://arxiv.org/abs/2302.04761>. Acesso em: 7 jun. 2026.

TAO, Dehao; MA, Guoliang; HUANG, Yongfeng; JIANG, Minghu. Membox: Weaving Topic Continuity into Long-Range Memory for LLM Agents, 2026. Disponível em: <https://arxiv.org/abs/2601.03785>. Acesso em: 9 jun. 2026.

VASWANI, Ashish; SHAZEER, Noam; PARMAR, Niki; USZKOREIT, Jakob; JONES, Llion; GOMEZ, Aidan N.; KAISER, Lukasz; POLOSUKHIN, Illia. Attention Is All You Need. In: ADVANCES in Neural Information Processing Systems. [S.l.: s.n.], 2017. Disponível em: <https://arxiv.org/abs/1706.03762>. Acesso em: 7 jun. 2026.

YAO, Shunyu; ZHAO, Jeffrey; YU, Dian; DU, Nan; SHAFRAN, Izhak; NARASIMHAN, Karthik; CAO, Yuan. ReAct: Synergizing Reasoning and Acting in Language Models. In: INTERNATIONAL Conference on Learning Representations.

[S.l.: s.n.], 2023. Disponível em: https://openreview.net/forum?id=WE_vluYUL-X. Acesso em: 7 jun. 2026.

ZHANG, Chao et al. TeaRAG: A Token-Efficient Agentic Retrieval-Augmented Generation Framework, 2025. Disponível em: <https://arxiv.org/abs/2511.05385>. Acesso em: 9 jun. 2026.

ZHENG, Yuanhang; LI, Peng; LIU, Wei; LIU, Yang; LUAN, Jian; WANG, Bin. ToolRerank: Adaptive and Hierarchy-Aware Reranking for Tool Retrieval. In: PROCEEDINGS of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation. [S.l.: s.n.], 2024. p. 16263–16273. Disponível em: <https://aclanthology.org/2024.lrec-main.1413/>. Acesso em: 9 jun. 2026.

ZHOU, Zijian; QU, Ao; WU, Zhaoxuan; KIM, Sunghwan; PRAKASH, Alok; RUS, Daniela; ZHAO, Jinhua; LOW, Bryan Kian Hsiang; LIANG, Paul Pu. MEM1: Learning to Synergize Memory and Reasoning for Efficient Long-Horizon Agents, 2025. Disponível em: <https://arxiv.org/abs/2506.15841>. Acesso em: 9 jun. 2026.

APÊNDICE A – DESCRIÇÃO 1

Textos elaborados pelo autor, a fim de completar a sua argumentação. Deve ser precedido da palavra APÊNDICE, identificada por letras maiúsculas consecutivas, travessão e pelo respectivo título. Utilizam-se letras maiúsculas dobradas quando esgotadas as letras do alfabeto.

ANEXO A – DESCRIÇÃO 2

São documentos não elaborados pelo autor que servem como fundamentação (mapas, leis, estatutos). Deve ser precedido da palavra ANEXO, identificada por letras maiúsculas consecutivas, travessão e pelo respectivo título. Utilizam-se letras maiúsculas dobradas quando esgotadas as letras do alfabeto.