



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

Breno Barroso Moreira

**Projeto e Avaliação de um Sistema de Memória para Agentes de
IA: Gerenciamento de Contexto e Persistência de Fatos entre Sessões**

Florianópolis
2026

Breno Barroso Moreira

**Projeto e Avaliação de um Sistema de Memória para Agentes de
IA: Gerenciamento de Contexto e Persistência de Fatos entre Sessões**

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(a). Jomi Fred Hübner, Dr(a).

Supervisor: Samir Pauli Fiates, Eng(a).

Florianópolis

2026

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

Moreira, Breno Barroso

Projeto e avaliação de um sistema de memória para agentes de IA : Gerenciamento de contexto e persistência de fatos entre sessões / Breno Barroso Moreira ; orientador, Jomi Fred Hübner, coorientador, Samir Pauli Fiates, 2026.

102 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. Sistemas de memória. 3. Agentes inteligentes. I. Hübner, Jomi Fred. II. Fiates, Samir Pauli. III. Universidade Federal de Santa Catarina. Graduação em Engenharia de Controle e Automação. IV. Título.

Breno Barroso Moreira

**Projeto e Avaliação de um Sistema de Memória para Agentes de
IA: Gerenciamento de Contexto e Persistência de Fatos entre Sessões**

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, 6 de julho de 2026.

Prof. Marcelo De Lellis Costa de Oliveira, Dr.
Coordenador do Curso

Banca Examinadora:



Documento assinado digitalmente

Jomi Fred Hübner

Data: 08/07/2026 07:52:21-0300

CPF: ***.334.769-**

Verifique as assinaturas em <https://v.ufsc.br>

Prof. Jomi Fred Hübner, Dr.

Orientador(a)

UFSC/CTC/EAS



Documento assinado digitalmente

SAMIR PAULI FIATES

Data: 07/07/2026 16:39:10-0300

Verifique em <https://validar.iti.gov.br>

Samir Pauli Fiates, Eng(a).

Supervisor(a)

HarboR Inteligência Industrial

Prof. Thiago Raulino Dal Pont, Dr.

Avaliador(a)

UFSC/CTC/PosAutomação

Prof. Hector Bessa Silveira, Dr.

Presidente da Banca

UFSC/CTC/EAS

Este trabalho é dedicado à minha família, em especial aos meus pais, avós e irmãos, que me moldaram em quem sou, sem vocês nada disso seria possível.

Também dedico a todos amigos que fiz durante a faculdade, em especial minha namorada e meus amigos de turma, vocês fizeram minha graduação mais leve e divertida.

AGRADECIMENTOS

Primeiramente, gostaria de agradecer aos meus pais, Israel e Patrícia, e aos meus avós, Eduardo, Lucy, Rui e Veraluz, por todo o incentivo que me foi dado ao longo da minha vida, seja educacional ou emocional, vocês foram o pilar da minha vida. Aos meus irmãos, Laura e Miguel, por todas as nossas brincadeiras durante esses anos.

Um agradecimento especial à minha namorada, Heloisa, que sempre me incentivou com palavras de apoio e esteve ao meu lado em todos os momentos. Foi essencial ter você comigo durante essa etapa.

Ao meu supervisor, Samir, que me acompanhou nessa jornada nos últimos meses. Por toda ajuda durante a escrita do projeto e ensinamentos que me foram dados, aprendi muitas coisas inovadoras durante o desenvolvimento desse trabalho. Também gostaria de agradecer ao meu orientador, Jomi, por ter aceito e me acompanhado durante o desenvolvimento.

Aos meus amigos da automação foram todos essenciais para o cumprimento dessa longa jornada na engenharia, em especial a turma 20.2 e a Arca. Estiveram comigo nos momentos de caos, de estudo e de descontração, que fazem parte da faculdade e são o que nos dá forças para continuar. Também gostaria de agradecer aos meus amigos da escola que dividiram esse momento da faculdade comigo, podemos não nos ver mais diariamente mas sei que posso contar com vocês.

Por fim, quero também agradecer a toda o curso de Automação, estudantes, egressos, professores e administrativos, em especial à ATACA, vocês que tornam o ambiente uma verdadeira família da automação, o que serve de apoio para muitos alunos, em momentos bons e ruins. Sempre terei orgulho de ter feito parte desse curso.

*“O desenho de nossas vidas, como a
chama da vela, é continuamente conduzido
em novas direções por diversos eventos
aleatórios que, juntamente com nossas
reações a eles, determinam nosso destino.”
(MLODINOW, 2009)*

DECLARAÇÃO DE PUBLICIDADE

Florianópolis, 7 de julho de 2026.

Na condição de representante da HarboR Inteligência Industrial 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
 **SAMIR PAULI FIATES**
Data: 07/07/2026 16:40:50-0300
Verifique em <https://validar.iti.gov.br>

Samir Pauli Fiates
HarboR Inteligência Industrial

RESUMO

Os Modelos de Linguagem de Grande Escala operam sem estado e não retêm informação entre chamadas, o que faz com que agentes de uso contínuo percam contexto, repitam erros, ignorem preferências do usuário e produzam respostas inconsistentes a cada nova sessão. Enviar todo o histórico a cada requisição não resolve o problema, pois eleva o custo por consulta, aumenta a latência e degrada a qualidade da resposta em contextos longos. Este trabalho projeta, desenvolve e avalia um sistema de memória de agente, aplicado à LivIA, o assistente de IA industrial da HarboR Inteligência Industrial. A solução divide a memória em dois módulos. O módulo de memória de sessão gerencia o contexto da interação corrente por meio de um buffer com janela deslizante, de um pipeline de compactação acionado por limite de mensagens ou de tokens, o que ocorrer primeiro, e de um sumário estruturado segmentado por mudança de tópico. O módulo de memória de longo prazo extrai preferências do usuário e fatos do ambiente operacional, persiste-os em banco relacional e vetorial indexado por usuário e empresa, e os recupera nas sessões seguintes injetando-os automaticamente no contexto. Um mecanismo de busca por ferramenta permite recuperar sob demanda resumos de sessões anteriores referenciados pelo operador. Uma política de admissão semântica baseada em distância de cosseno e em um árbitro baseado em modelo de linguagem evita duplicatas e resolve contradições entre memórias, com limiares calibrados sobre um conjunto de dados sintético. A implementação usa um *framework* de orquestração de agentes, banco de dados em memória para o contexto ativo, banco relacional e vetorial para o armazenamento durável e um modelo de linguagem comercial para sumarização, extração e geração de respostas. O sistema preserva a continuidade e a personalização entre sessões, isola estritamente as memórias por usuário e empresa e mantém o histórico de conversa dentro do limite configurado independentemente da duração da sessão. A avaliação verificou o atendimento de todos os requisitos funcionais e confirmou que a memória ativa não degrada a latência mediana de resposta, mantendo-a equivalente à configuração sem memória.

Palavras-chave: Agentes de inteligência artificial. Memória de agente. Gerenciamento de contexto. Modelos de linguagem.

ABSTRACT

Large Language Models operate in a stateless manner and retain no information between calls, causing continuously-used agents to lose context, repeat errors, ignore user preferences, and produce inconsistent responses across sessions. Sending the full conversation history with each request does not solve the problem, as it raises per-query cost, increases latency, and degrades response quality in long contexts. This work designs, develops, and evaluates an agent memory system, applied to LivIA, the industrial AI assistant developed by HarboR Inteligência Industrial. The solution divides memory into two modules. The session memory module manages the current interaction context through a sliding-window buffer, a compaction pipeline triggered by a message-count or token limit, whichever comes first, and a structured summary segmented by topic change. The long-term memory module extracts user preferences and operational environment facts, persists them in a relational and vector database indexed by user and company, and retrieves them in subsequent sessions injecting them automatically into context. A tool-based search mechanism allows on-demand retrieval of previous session summaries referenced by the operator. A semantic admission policy based on cosine distance and a language-model judge prevents duplicate memories and resolves contradictions, with thresholds calibrated on a synthetic dataset. The implementation uses an agent orchestration framework, an in-memory database for active context, a relational and vector database for durable storage, and a commercial language model for summarization, extraction, and response generation. The system preserves continuity and personalization across sessions, strictly isolates memories by user and company, and keeps the conversation history within the configured limit regardless of session length. The evaluation verified that all functional requirements were met and confirmed that active memory does not degrade median response latency, keeping it equivalent to the baseline configuration.

Keywords: Artificial intelligence agents. Agent memory. Context management. Language models.

LISTA DE FIGURAS

Figura 1 – Evolução esquemática da inteligência artificial.	23
Figura 2 – Exemplo de tokenização de uma frase.	26
Figura 3 – Exemplo de sequência de mensagens enviada ao modelo em uma interação <i>multi-turn</i>	27
Figura 4 – Comparação conceitual entre memória de agente, memória de LLM, Recuperação Aumentada por Geração (<i>Retrieval-Augmented Generation</i>) (RAG) e engenharia de contexto.	29
Figura 5 – Fluxo de funcionamento do RAG.	30
Figura 6 – Representação plana de memória.	33
Figura 7 – Representação planar de memória.	34
Figura 8 – Três funções de memória em agentes de IA.	35
Figura 9 – Dinâmicas do ciclo de vida da memória do agente.	39
Figura 10 – Projeção simplificada de três frases no espaço de <i>embedding</i>	44
Figura 11 – Desempenho em função da posição da informação relevante no contexto.	47
Figura 12 – Zonas de desempenho em função do uso da janela de contexto.	48
Figura 13 – Arquitetura geral em camadas do agente.	52
Figura 14 – Fluxo de um turno: decisão de contexto e envio da resposta.	53
Figura 15 – Caminho assíncrono de memória após o envio da resposta.	54
Figura 16 – Buffer de mensagens com janela deslizante.	55
Figura 17 – Fluxo do pipeline de compactação e segmentação por tópico.	59
Figura 18 – Zonas de decisão da política de admissão de memória.	65
Figura 19 – Fluxo de decisão para persistência do registro de sessão.	73
Figura 20 – Distribuição das distâncias de cosseno por classe e zonas de admissão resultantes da calibração.	80
Figura 21 – Operador pergunta sobre conversa anterior e o agente recupera o resumo correto a partir do banco.	84
Figura 22 – Trace completo no Langfuse com <i>input</i> e <i>output</i> da interação.	85
Figura 23 – Chamada à ferramenta <i>search_session_history</i> : parâmetros enviados pelo agente e registros retornados pelo banco com <i>top_k</i>	86
Figura 24 – Operador informa papel e setor ao agente, ativando o padrão de detecção do extrator.	87
Figura 25 – Dois traces no Langfuse: a mensagem do operador e o LLM extrator acionado de forma assíncrona.	87
Figura 26 – <i>System prompt</i> do extrator com o contexto de ambiente como entrada.	88
Figura 27 – Saída do extrator: setor de Injeção e função de supervisor salvos como dois registros distintos.	88

Figura 28 – Nova sessão com cache limpo: agente responde com o OEE da Injetora sem que o operador informe o setor.	88
Figura 29 – <i>System prompt</i> da nova sessão com o bloco [ENVIRONMENT FACTS] injetado a partir dos fatos persistidos.	89
Figura 30 – Operador declara preferência de formato de resposta ao agente. . .	90
Figura 31 – Saída do extrator com a preferência de formato identificada e estruturada.	90
Figura 32 – Nova sessão: agente aplica a preferência de formato sem que o operador precise repeti-la.	91
Figura 33 – Distribuição de latência por turno (COM vs SEM, $n = 36$ por condição). .	92
Figura 34 – Custo médio por turno (COM vs SEM), por rodada.	93

LISTA DE QUADROS

Quadro 1 – Requisitos funcionais do sistema.	50
Quadro 2 – Requisitos não-funcionais do sistema.	51
Quadro 3 – Estrutura do sumário estruturado de sessão em Linguagem de Marcação Extensível (<i>eXtensible Markup Language</i>) (XML).	56
Quadro 4 – Estado da tabela após a primeira compactação (<i>continuous</i>).	58
Quadro 5 – Estado da tabela após a segunda compactação (<i>continuous</i> com <i>upsert</i>).	58
Quadro 6 – Estado da tabela após a terceira compactação (<i>discontinuous</i> com <i>insert</i>).	58
Quadro 7 – Formato de representação das memórias de longo prazo.	61
Quadro 8 – Exemplos de representação de preferências do usuário e fatos do ambiente.	61
Quadro 9 – Componentes tecnológicos do sistema LIVIA.	66
Quadro 10 – Parâmetros configuráveis do sistema de memória.	67
Quadro 11 – Estrutura do prompt de sumarização de sessão.	70
Quadro 12 – Exemplo de sumário estruturado gerado pelo sumarizador.	71
Quadro 13 – Campos da tabela <i>session_snapshots</i>	73
Quadro 14 – Prompt do extrator unificado de memória de longo prazo.	76
Quadro 15 – Campos das tabelas <i>environment_facts</i> e <i>user_preferences</i>	77
Quadro 16 – Esquema relacional compartilhado pelas tabelas <i>environment_facts</i> e <i>user_preferences</i>	78
Quadro 17 – Zonas de decisão da política de admissão A-MAC.	79
Quadro 18 – Avaliação dos requisitos funcionais e não funcionais.	83

LISTA DE TABELAS

Tabela 1 – Projeção da componente de histórico de conversa por turno (estimativa por extrapolação).	94
---	----

LISTA DE ABREVIATURAS E SIGLAS

CI	Integração Contínua (<i>Continuous Integration</i>)
HNSW	Mundo Pequeno Navegável Hierárquico (<i>Hierarchical Navigable Small World</i>)
IA	Inteligência Artificial
LLM	Modelo de Linguagem de Grande Escala (<i>Large Language Model</i>)
MCP	Protocolo de Contexto de Modelo (<i>Model Context Protocol</i>)
MES	Sistema de Execução da Manufatura (<i>Manufacturing Execution System</i>)
OEE	Eficácia Geral do Equipamento (<i>Overall Equipment Effectiveness</i>)
RAG	Recuperação Aumentada por Geração (<i>Retrieval-Augmented Generation</i>)
XML	Linguagem de Marcação Extensível (<i>eXtensible Markup Language</i>)

SUMÁRIO

1	INTRODUÇÃO	18
1.1	A EMPRESA	19
1.2	OBJETIVOS	20
1.3	ORGANIZAÇÃO DO DOCUMENTO	21
2	FUNDAMENTAÇÃO TEÓRICA	22
2.1	CONCEITOS FUNDAMENTAIS	23
2.1.1	Inteligência Artificial	23
2.1.2	Modelos de Linguagem de Grande Escala e Agentes Inteligentes	24
2.1.3	Tokens e Janela de contexto	25
2.1.4	Memória de agente	27
2.2	FRONTEIRAS DE MEMÓRIA DE AGENTE	28
2.2.1	Memória de LLM	29
2.2.2	RAG	30
2.2.3	Engenharia de Contexto	31
2.3	FORMAS DE REPRESENTAÇÃO DE MEMÓRIA	32
2.3.1	Flat	32
2.3.2	Planar	33
2.3.3	Hierárquica	34
2.4	FUNÇÕES DE MEMÓRIA EM AGENTES	34
2.4.1	Factual	35
2.4.1.1	Memória Factual do Usuário	36
2.4.1.2	Memória Factual do Ambiente	36
2.4.2	Trabalho	37
2.4.2.1	Turno único	37
2.4.2.2	Multi-turno	37
2.4.3	Experiencial	38
2.5	DINÂMICAS DO CICLO DE VIDA DA MEMÓRIA	38
2.5.1	Formação	39
2.5.1.1	Sumarização Semântica	39
2.5.1.2	Destilação de Conhecimento	40
2.5.1.3	Construção Estruturada	40
2.5.1.4	Representação Latente	41
2.5.1.5	Internalização Paramétrica	41
2.5.2	Evolução	41
2.5.2.1	Consolidação	41
2.5.2.2	Atualização	42
2.5.2.3	Esquecimento	42

2.5.3	Recuperação	42
2.5.3.1	Tempo e Intenção	42
2.5.3.2	Construção da Consulta	43
2.5.3.3	Estratégias de Busca	43
2.5.3.4	Pós-Processamento	44
3	DESCRIÇÃO DO PROBLEMA E REQUISITOS TÉCNICOS	45
3.1	CONTEXTO E MOTIVAÇÃO	45
3.2	DEFINIÇÃO DO PROBLEMA	46
3.3	REQUISITOS TÉCNICOS	49
3.3.1	Requisitos Funcionais	49
3.3.2	Requisitos Não-Funcionais	50
4	SOLUÇÃO PROPOSTA E METODOLOGIA UTILIZADA	52
4.1	VISÃO GERAL DA ARQUITETURA DO SISTEMA	52
4.2	MÓDULO DE MEMÓRIA DE SESSÃO	54
4.2.1	Pipeline de compactação do histórico	54
4.2.2	Sumário Estruturado de Sessão	56
4.2.3	Controle de Continuidade de Tópico	57
4.3	MÓDULO DE MEMÓRIA DE LONGO PRAZO	60
4.3.1	Preferências do Usuário e Fatos do Ambiente	60
4.3.2	Destilação Estruturada	62
4.3.3	Persistência e Recuperação Híbrida	63
4.3.4	Controle de Admissão de Memória	64
5	DESENVOLVIMENTO E ANÁLISE DOS RESULTADOS OBTIDOS	66
5.1	IMPLEMENTAÇÃO DO AMBIENTE E INFRAESTRUTURA	66
5.2	DESENVOLVIMENTO DO MÓDULO DE MEMÓRIA DE SESSÃO	67
5.2.1	Pipeline de Compactação do Histórico	68
5.2.2	Sumário Estruturado de Sessão	68
5.2.3	Controle de Continuidade de Tópico e Persistência do Registro de Sessão	71
5.2.4	Recuperação de Sessão	74
5.3	DESENVOLVIMENTO DO MÓDULO DE MEMÓRIA DE LONGO PRAZO	74
5.3.1	Extração Unificada de Memória	75
5.3.2	Preferências do Usuário e Fatos do Ambiente	77
5.3.3	Controle de Admissão de Memória	78
5.3.4	Corte de Memória e Controle de Compactação	80
5.3.5	Recuperação Híbrida	81
5.4	AVALIAÇÃO DE DESEMPENHO	81
5.4.1	Avaliação por Requisitos Funcionais e Não Funcionais	82
5.4.2	Exemplos de Operação	84

5.4.2.1	Recuperação de Histórico de Sessão	84
5.4.2.2	Extração e Injeção de Fatos do Ambiente	86
5.4.2.3	Extração e Injeção de Preferências do Usuário	89
5.4.3	Avaliação de Eficiência e Custo	91
6	CONCLUSÃO	95
6.1	OBJETIVOS ATINGIDOS	95
6.2	LIMITAÇÕES ENCONTRADAS	96
6.3	IMPACTOS	96
6.4	TRABALHOS FUTUROS	97
	REFERÊNCIAS	99

1 INTRODUÇÃO

Com o grande crescimento e evolução da Inteligência Artificial (IA) nos últimos anos, os Modelos de Linguagem de Grande Escala (*Large Language Models* – LLMs) expandiram-se significativamente no ambiente corporativo, deixando de atender apenas casos de atendimento básico ao cliente para também auxiliar operadores na execução de fluxos de trabalho e na resolução de problemas práticos do dia a dia.

Por esse motivo, também aparecem novos desafios relacionados ao modo como esses modelos funcionam. Os LLMs não possuem estado (*stateless*) por natureza: o modelo, durante a inferência, não mantém uma memória entre chamadas. Sem o re-envio do histórico anterior, um LLM perde o contexto da conversa e não consegue acessar informações discutidas previamente. Neste trabalho, entende-se por contexto o conjunto de informações fornecidas ao modelo em uma chamada, incluindo instruções do sistema, histórico da conversa e dados externos. Assim, o histórico recente, preferências do usuário, decisões tomadas em uma sessão e dados específicos da empresa precisam ser enviados novamente a cada interação com o modelo.

O funcionamento de um LLM baseia-se na previsão probabilística do próximo token (BROWN *et al.*, 2020). Este cenário faz o modelo gerar a sequência textual com base nas informações disponíveis no texto antecedente. Quando informações concretas não estão presentes no contexto, o modelo tende a preencher lacunas com informações estatisticamente prováveis. Se essas informações são falsas, imprecisas ou sem aderência ao caso real, o fenômeno é normalmente tratado como alucinação. Em agentes utilizados ao longo de múltiplas interações, esse comportamento pode fazer com que o sistema repita erros, ignore preferências do usuário, perca aprendizados de sessões anteriores e produza respostas pouco confiáveis para decisões operacionais.

Uma forma direta de tentar resolver o problema seria enviar todo o histórico da interação para o LLM, ou injetar no *prompt* do agente todas as informações possivelmente relevantes. Com isso, o modelo receberia mais dados para gerar a resposta. Entretanto, essa estratégia esbarra em limitações práticas importantes. O texto de entrada é processado em unidades chamadas *tokens*, e cada modelo possui uma janela de contexto com limite físico e computacional. Como as APIs dos principais modelos (OpenAI, Anthropic, Google) comerciais cobram pela quantidade de *tokens* processados, contextos extensos aumentam o custo de cada chamada. Além disso, quanto maior o contexto enviado, maior tende a ser a latência para geração da resposta, o que pode tornar a solução pouco adequada para o uso operacional contínuo. Somado a isso, cada modelo impõe um limite físico de tokens que o contexto pode conter, como o Claude Opus, que suporta até 1 milhão de tokens (ANTHROPIC, 2026), e ao atingir esse limite, mensagens antigas precisam ser descartadas.

Além do custo, há uma limitação de qualidade. O fenômeno conhecido como *Lost in the Middle* mostra que modelos de linguagem tendem a utilizar melhor informações posicionadas no início ou no final do contexto, enquanto a acurácia cai quando a informação relevante está localizada no meio de contextos longos. Liu *et al.* (2023) observam esse comportamento como uma curva em U: o desempenho é maior nas extremidades do contexto e degrada significativamente na região central. Isso significa que apenas aumentar a quantidade de informação enviada ao modelo não garante que a informação correta será efetivamente usada.

Outro problema é que o contexto pode se perder ao longo do tempo mesmo quando a informação relevante está disponível. Yufeng Du *et al.* (2025) mostram que o aumento do comprimento da entrada, por si só, pode prejudicar o desempenho de LLMs em tarefas de raciocínio, mesmo quando o modelo consegue recuperar perfeitamente as evidências necessárias. Esse resultado reforça que a dificuldade não está apenas em encontrar a informação dentro de um contexto longo; o próprio tamanho do contexto pode reduzir a capacidade do modelo de usar essa informação de forma adequada.

Portanto, não basta preencher a janela de contexto com todo o histórico disponível. É necessário possuir um bom gerenciamento de memória e conhecimento externo, capaz de selecionar, compactar, persistir e recuperar as informações corretas antes da geração da resposta. Conforme discutido por Pengfei Du (2026), a memória permite persistir, organizar e recuperar seletivamente informações ao longo de interações, transformando um gerador de texto sem estado em um agente capaz de lembrar interações anteriores, recuperar conhecimento relevante e adaptar suas respostas ao histórico de uso. Essa capacidade é particularmente relevante em aplicações nas quais a veracidade da resposta influencia custos, produtividade e segurança das decisões operacionais.

Este Projeto de Fim de Curso trata da adição de memória persistente a um agente. A proposta combina uma Memória de Sessão (*Session Memory*), entendida neste trabalho como a memória de curto prazo (*Short-Term Memory*) responsável pelo gerenciamento do contexto durante uma interação, com uma camada de Memória de Longo Prazo (*Long-Term Memory*). Essa organização permite que o agente mantenha uma janela de contexto mais limpa e coerente ao longo das interações. Com isso, o sistema busca preservar informações relevantes entre sessões, como preferências do usuário e fatos do ambiente operacional, sem depender do envio integral do histórico a cada nova requisição.

1.1 A EMPRESA

A HarboR Inteligência Industrial, com sede em Florianópolis, Santa Catarina, é uma empresa de engenharia que desenvolve e integra sistemas industriais para o

chão de fábrica (HARBOR INTELIGÊNCIA INDUSTRIAL, 2026b). Fundada em Santa Catarina, a empresa nasceu voltada à prestação de serviços de desenvolvimento de software para sistemas MES (*Manufacturing Execution Systems*) customizados e, ao longo de sua trajetória, consolidou sua atuação em soluções para digitalização, monitoramento e controle de processos industriais. Em 2026, a HarboR completa 30 anos de atuação no setor industrial.

Entre as soluções desenvolvidas pela empresa está o LiveMES, sistema MES criado para apresentar, de forma automática e em tempo real, o que está acontecendo no chão de fábrica (HARBOR INTELIGÊNCIA INDUSTRIAL, 2026a). O sistema permite o monitoramento da produção, controle de OEE (*Overall Equipment Effectiveness*) por máquina ou linha, gestão de ordens de produção, visualização de indicadores e integração com sistemas de gestão e automação. Dessa forma, o LiveMES busca transformar dados produtivos em informações acessíveis para operadores, gestores e demais usuários envolvidos na rotina da fábrica.

O presente trabalho foi desenvolvido no time de Desenvolvimento e IA da HarboR, no contexto da criação da LivIA, um *chatbot* associado ao LiveMES e à aplicação de Inteligência Artificial (IA) no ambiente industrial. A proposta da LivIA é permitir que operadores e usuários do software consultem informações sobre o andamento da fábrica por meio de linguagem natural, além de outras funcionalidades que poderão ser incorporadas ao produto. Nesse projeto, o autor participou ativamente da construção do *chatbot* desde suas etapas iniciais, com foco no desenvolvimento de mecanismos de memória capazes de manter contexto, preferências e aprendizados ao longo das interações.

1.2 OBJETIVOS

O objetivo geral deste trabalho é projetar, desenvolver e avaliar tecnicamente um sistema de memória para um agente de IA, contemplando o gerenciamento de contexto de curto prazo e a persistência de memórias de longo prazo em bancos de dados relacionais e vetoriais. Busca-se, com isso, melhorar a continuidade entre interações, reduzir custos associados ao envio de contextos extensos e mitigar a ocorrência de respostas inconsistentes.

Os objetivos específicos deste trabalho são:

- Revisar e mapear a literatura para poder estruturar a fundamentação teórica sobre sistemas de memória para agentes de IA.
- Desenvolver o módulo de Memória de Sessão, implementando um *pipeline* para compactação do histórico utilizando diferentes técnicas para gerenciar o tamanho do contexto de curto prazo dentro de um limite de tokens.

- Desenvolver o módulo de Memória de Longo Prazo, com extração e persistência de preferências do usuário e fatos do ambiente operacional, indexados por usuário e empresa, para recuperação nas sessões seguintes.
- Realizar uma avaliação de desempenho para calcular a eficácia e veracidade das respostas.

1.3 ORGANIZAÇÃO DO DOCUMENTO

O presente documento está organizado da seguinte maneira. O Capítulo 2 apresenta a fundamentação teórica necessária para o entendimento do trabalho, incluindo conceitos de IA, LLMs, agentes inteligentes, janela de contexto e memória de agente. O Capítulo 3 descreve o problema tratado e os requisitos técnicos considerados no projeto. O Capítulo 4 apresenta a solução proposta e a metodologia utilizada, detalhando a arquitetura do sistema e os módulos de memória de sessão e de longo prazo. O Capítulo 5 descreve o desenvolvimento realizado e a análise dos resultados obtidos. Por fim, o Capítulo 6 apresenta as conclusões, as limitações encontradas, os impactos do trabalho e sugestões de trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo enquadra toda a fundamentação necessária para o entendimento das funcionalidades e conceitos que aparecerão no decorrer do projeto.

A Seção 2.1 tratará as noções essenciais sobre inteligência artificial moderna, dos modelos de linguagem de grande escala e dos agentes inteligentes, além de introduzir a caracterização de memória de agente. Em seguida, é apresentada uma comparação entre os tipos de memória entre referenciais teóricos relacionados na Seção 2.2, como memória de LLMs, Recuperação Aumentada por Geração (*Retrieval-Augmented Generation* – RAG) e engenharia de contexto (*Context Engineering*).

As seções seguintes concentram-se especificamente na memória do agente, seguindo a organização conceitual apresentada por Hu *et al.* (2026). A Seção 2.3 discute sobre as unidades representacionais que a memória pode desempenhar. Os diferentes tipos de informações que uma memória pode assumir são tratados na Seção 2.4, que avalia as funções que elas têm baseado em sua informação. A Seção 2.5 trata do ciclo de vida da memória desde sua formação, evolução e por fim a recuperação dela em um contexto necessário.

2.1 CONCEITOS FUNDAMENTAIS

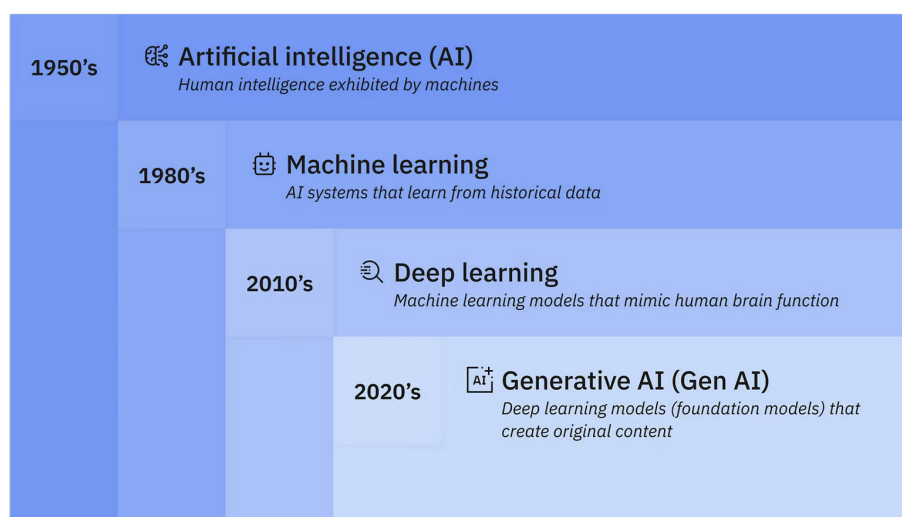
Nesta seção, serão tratados os conceitos fundamentais necessários para entender o contexto em que a memória de agente é aplicada. Para isso, são apresentados inicialmente as noções básicas de inteligência artificial, modelos de linguagem de grande escala e agentes inteligentes. Em seguida, são discutidos o que são tokens e janela de contexto, que ajudam a compreender as limitações dos modelos e a necessidade de mecanismos externos de memória.

2.1.1 Inteligência Artificial

Entende-se por IA o desenvolvimento de sistemas para execução de tarefas humanas. Dentro de suas múltiplas áreas, este trabalho delimita-se ao contexto de sistemas de IA generativa para a criação de agentes. Segundo Russell e Norvig (2010), a IA é uma das áreas mais recentes nas áreas de ciência e engenharia, tendo seu desenvolvimento consolidado a partir da segunda metade do século XX.

Desde então, a área passou por muitas mudanças, e essa evolução pode ser apresentada, de forma esquemática, pela transição entre: sistemas clássicos, aprendizado de máquina, aprendizado profundo, e mais recentemente, a IA generativa, conforme ilustrado na Figura 1. Toda essa trajetória mostra a passagem de sistemas rigidamente programados para modelos capazes de aprender padrões a partir de dados e gerar respostas e conteúdos em linguagem natural.

Figura 1 – Evolução esquemática da inteligência artificial.



Fonte: IBM (2026a).

Além da perspectiva histórica, a IA pode ser definida a partir de diferentes formas. Russell e Norvig (2010) organizam essas definições em quatro categorias

principais: sistemas que pensam como humanos, sistemas que pensam racionalmente, sistemas que agem como humanos e sistemas que agem racionalmente. Esses conceitos mostram que a IA não se limita apenas à reprodução de capacidades humanas, mas também envolvem sistemas que são capazes de selecionar ações em função de objetivos, informações e racionalidade.

Neste trabalho, vamos focar na categoria de sistemas que agem racionalmente, pois ela se conecta diretamente ao agente inteligente. A partir dessa perspectiva, a IA deixa de ser apenas um conjunto de técnicas de processamento ou geração de informação e passa a ser parte de um sistema capaz de perceber uma situação, processar informações do contexto e tomar uma decisão em função do seu objetivo.

2.1.2 Modelos de Linguagem de Grande Escala e Agentes Inteligentes

Zhao *et al.* (2023) descrevem LLMs como modelos de linguagem pré-treinados em larga escala, geralmente baseados em arquiteturas *Transformer*, capazes de apresentar forte desempenho em tarefas de compreensão e geração de linguagem natural. A IBM também descreve LLMs como modelos de *deep learning* treinados em grandes quantidades de dados, capazes de gerar e entender a linguagem natural, um LLM consegue gerar diferentes tipos de informações, desde um texto para um e-mail até um código de linguagem de programação (IBM, 2026b). Eles funcionam como grandes modelos estatísticos, capazes de prever o próximo token baseado em seu contexto anterior (BROWN *et al.*, 2020). Por se tratarem de modelos não determinísticos, os LLMs podem gerar diversas respostas para a mesma pergunta.

Eles funcionam da seguinte forma, recebem uma entrada textual, chamada de *prompt*. Através dessas informações, com seu grande número de parâmetros os LLMs geram estatisticamente a resposta baseada no contexto enviado, gerando sequencialmente as informações. Em muitos LLMs modernos, esse processamento é realizado por arquiteturas baseadas em *Transformers*, que utilizam mecanismos de atenção para representar relações entre diferentes partes da sequência (VASWANI *et al.*, 2017). Os grandes modelos não possuem estado. Isto é, durante a inferência da resposta, agem apenas em cima do que foi enviado. Eles não possuem nenhuma forma de memória persistente com informações anteriores ou externas, apenas as que foram utilizadas para seu treinamento (DU, P., 2026).

A definição de agente inteligente é anterior aos LLMs e pode ser entendida, de forma geral, como um sistema que percebe o ambiente e age em função de objetivos (RUSSELL; NORVIG, 2010). Antes dos LLMs, esses agentes já eram construídos com outras abordagens, como busca, planejamento, sistemas baseados em regras e aprendizado por reforço (RUSSELL; NORVIG, 2010). O modelo de linguagem é apenas uma das formas de implementar o raciocínio do agente, e este trabalho recorta o caso dos agentes baseados em LLM, em que o modelo passou a ser utilizado como

componente central da arquitetura (WANG *et al.*, 2024). Um agente baseado em IA não produz apenas respostas isoladas, ele opera em diferentes ciclos onde percebe, decide e age em um determinado contexto. Essas ideias fornecem a fundamentação necessária para o design de *prompts*, permitindo que o agente interprete entradas, utilize o contexto de forma efetiva, consulte ferramentas e produza respostas adequadas a tarefa requisitada. O uso de ferramentas externas em conjunto com o raciocínio gerado por LLMs é explorado, por exemplo, em abordagens como ReAct (raciocínio e ação, do inglês *Reasoning and Acting*) (YAO *et al.*, 2022).

Manter estado entre interações é um problema antigo no estudo de agentes, tratado pelas arquiteturas clássicas. Embora os LLMs sejam o núcleo dos agentes baseados em modelos de linguagem, seu funcionamento individual não assegura a persistência temporal entre conversas. Portanto, é indispensável integrar uma ferramenta que proporcione memória persistente ao agente, permitindo a continuidade das informações.

2.1.3 Tokens e Janela de contexto

Antes de ser processada por um LLM, a entrada textual é convertida em *tokens* por meio de um processo chamado tokenização. Esse processo transforma o conteúdo enviado em unidades menores de texto, chamadas de *tokens*, que constituem a unidade básica de processamento interpretada pelo LLM (BALARABE, 2025; OPENAI, 2026). Esses *tokens* podem representar palavras, subpalavras, sinais de pontuação, espaços ou caracteres, dependendo do tokenizador utilizado. A tokenização em subpalavras, por exemplo, é uma técnica amplamente utilizada para lidar com vocabulários abertos e palavras raras em modelos de linguagem (SENNRICH; HADDOW; BIRCH, 2016). Cada token é associado a um identificador numérico no vocabulário do tokenizador, que posteriormente é convertido em representações vetoriais internas utilizadas pelo modelo. A Figura 2 ilustra esse processo aplicado a uma frase de exemplo.

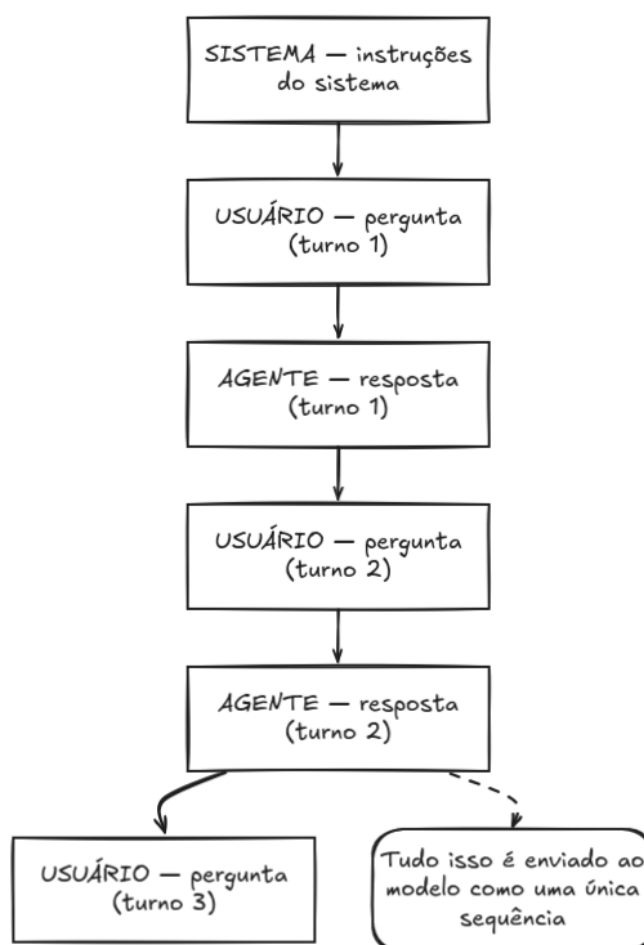
Figura 2 – Exemplo de tokenização de uma frase.



Fonte: Captura de tela de OpenAI (2026).

Define-se a janela de contexto como o limite máximo de tokens que o modelo consegue considerar para gerar uma resposta (ANTHROPIC, 2026). Esse limite não inclui apenas do texto enviado pelo usuário, esse conjunto de tokens pode conter instruções do sistema, histórico da conversa, documentos ou informações inseridas no contexto e em algumas situações os próprios tokens de saída. Em interações *multi-turn*, a janela de contexto pode incluir também o histórico da interação do usuário com o agente. Como consequência disso, a cada turno com o usuário esse histórico se acumula até que o limite da janela seja atingido. Esse gargalo tem motivado o desenvolvimento de janelas de contexto cada vez maiores em modelos recentes, como os modelos Claude com janela de contexto de até 1M de tokens documentados pela Anthropic (ANTHROPIC, 2026). A Figura 3 ilustra como esses elementos se organizam em uma única sequência enviada ao modelo a cada turno, incluindo as instruções do sistema e o histórico acumulado da conversa.

Figura 3 – Exemplo de sequência de mensagens enviada ao modelo em uma interação *multi-turn*.



Fonte: Elaborado pelo autor.

Contudo, mesmo com uma grande janela de contexto, Yufeng Du *et al.* (2025) mostram que muitos tokens dividindo a atenção do modelo prejudicam a resposta. Mesmo que o modelo consiga recuperar informações relevantes durante turnos de conversa, sua performance decai substancialmente conforme cresce seu *input* (entrada). Outro efeito já documentado se chama *Lost in the Middle* (LIU *et al.*, 2023). Quanto maior a quantidade de informações enviadas a um modelo, maior a importância da posição das informações no prompt. Dando peso maior para informações no começo e depois no fim do contexto. Perdendo informações que ficam no meio e gerando uma forma de U em relevância de contexto.

2.1.4 Memória de agente

A memória de agente (*agent memory*) é necessária dadas as limitações já vistas devido aos LLMs e também pela necessidade do agente de tomar decisões com mais informações do que apenas a observação atual. Como o histórico de interações

influencia diretamente a qualidade das decisões do agente, a observação instantânea nem sempre é suficiente para tomar boas decisões (HU *et al.*, 2026). Ele precisa usar informações de interações anteriores, dentro da tarefa atual ou de tarefas passadas, para poder entregar um resultado melhor e com mais eficácia a dado contexto. Com a memória, o agente pode manter e recuperar informações em memória de curto prazo (*short-term memory*), relacionada à tarefa ou interação atual, e memória de longo prazo (*long-term memory*), relacionada a tarefas ou sessões anteriores, como explorado em sistemas de memória para conversas longas e interações sustentadas (PACKER *et al.*, 2023; ZHONG *et al.*, 2024).

O sistema externo ou um componente arquitetural que armazena, organiza e permite que o agente possa posteriormente recuperar informações é o que chamamos de memória. Com esses dados, ele pode buscar informações relevantes para apoiar a continuidade, personalização ou decisões com base no que ele sabe sobre o ambiente ou usuário (DU, P., 2026; ZHONG *et al.*, 2024; PARK *et al.*, 2023).

A memória pode ser representada de diferentes formas. Como um *buffer* textual, um banco de chave-valor (key-value), bancos de dados relacionais, bancos de dados vetoriais, um grafo ou uma estrutura híbrida (HU *et al.*, 2026). O MemGPT, por exemplo, propõe um gerenciamento de memória inspirado em sistemas operacionais para lidar com diferentes camadas de contexto e memória externa (PACKER *et al.*, 2023).

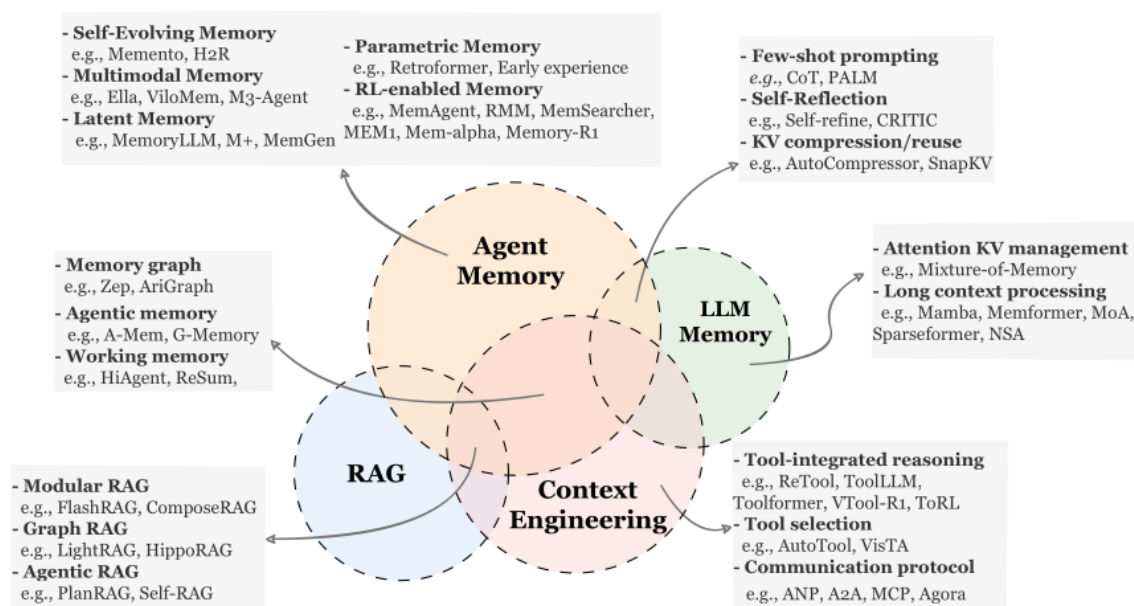
A memória não é estática, ela possui um ciclo de vida dinâmico. Começa em sua formação, atualiza ou evolui e depois é recuperada quando relevante (HU *et al.*, 2026).

2.2 FRONTEIRAS DE MEMÓRIA DE AGENTE

Esta seção aborda as fronteiras entre memória de agente e outros métodos próximos, como memória de LLM, Recuperação Aumentada por Geração (*Retrieval-Augmented Generation* – RAG) e engenharia de contexto (*Context Engineering*). Embora esses conceitos possam utilizar técnicas semelhantes, como recuperação de informação, compressão de contexto e uso de dados externos, isso não significa que eles sejam iguais. Essa proximidade entre técnicas pode gerar confusão na distinção do que realmente caracteriza a memória de agente.

A Figura 4 apresenta uma visão geral dessas relações, mostrando as áreas de sobreposição entre cada abordagem e os pontos em que elas se diferenciam. A partir dessa comparação, as subseções seguintes discutem brevemente as fronteiras entre memória de agente e memória de LLM, RAG e engenharia de contexto.

Figura 4 – Comparação conceitual entre memória de agente, memória de LLM, RAG e engenharia de contexto.



Fonte: Hu *et al.* (2026).

2.2.1 Memória de LLM

A memória de LLM está relacionada às formas pelas quais o próprio modelo armazena, acessa ou utiliza informações. Isso pode envolver o conhecimento aprendido durante o treinamento, armazenado de forma implícita nos parâmetros do modelo, ou mecanismos internos e arquiteturais voltados ao uso mais eficiente do contexto, como atenção, cache de chaves e valores e modelos com janelas de contexto maiores (HU *et al.*, 2026; DU, P., 2026).

Apesar de também serem chamadas de memória, essas abordagens não possuem o mesmo papel da memória de agente. A memória paramétrica do LLM é estática após o treinamento e não é atualizada a cada interação comum com o usuário. Já a janela de contexto permite que o modelo utilize informações fornecidas na entrada, mas essas informações são transitórias e deixam de estar disponíveis quando não são reenviadas ao modelo. Dessa forma, a memória de LLM está mais ligada à capacidade interna do modelo de representar e processar informações, enquanto a memória de agente está relacionada à persistência, organização, atualização e recuperação de informações ao longo do tempo.

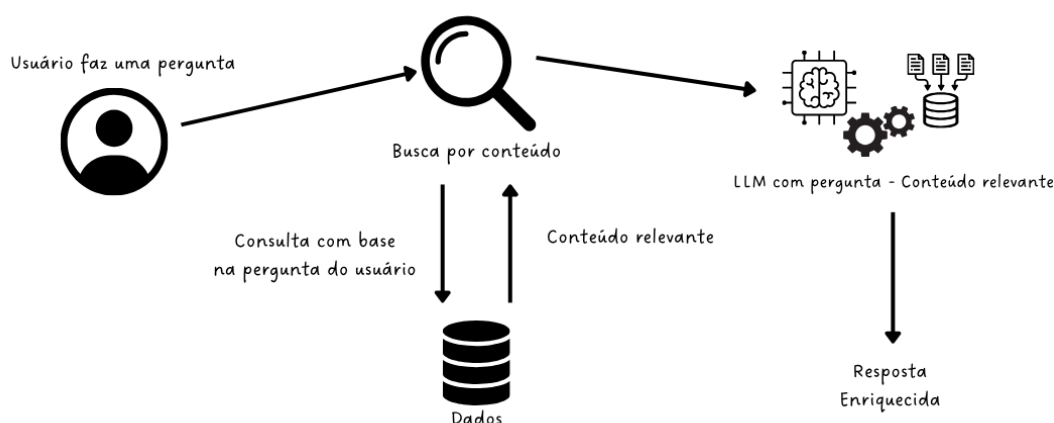
Assim, a principal diferença está no ciclo de vida da informação. Na memória de LLM, o conhecimento está incorporado aos parâmetros ou limitado ao contexto imediato da inferência. Na memória de agente, as informações podem ser registradas, avaliadas, modificadas e recuperadas posteriormente para apoiar decisões futuras. Por

isso, embora a memória de agente possa utilizar o LLM para interpretar, resumir ou recuperar informações, ela não se confunde com a memória interna do modelo.

2.2.2 RAG

A Recuperação Aumentada por Geração (*Retrieval-Augmented Generation* – RAG) é uma abordagem que combina o LLM com um mecanismo de busca em uma base de conhecimento externa. Em vez de depender apenas das informações aprendidas durante o treinamento, o modelo recebe documentos, trechos ou registros relevantes recuperados antes da geração da resposta. Com isso, o RAG permite que a resposta seja gerada com base em informações externas, específicas ou mais atualizadas do que aquelas presentes nos parâmetros do modelo. A Figura 5 ilustra esse fluxo.

Figura 5 – Fluxo de funcionamento do RAG.



Fonte: Adaptado de Redação StartSe (2024).

Essa abordagem possui uma proximidade importante com a memória de agente, pois também utiliza armazenamento, indexação e recuperação de informações. Em muitos casos, tanto sistemas de RAG quanto sistemas de memória podem usar bancos vetoriais, bases documentais, grafos de conhecimento ou busca semântica para encontrar informações relevantes. Por exemplo, estruturas baseadas em grafos podem ser utilizadas tanto em abordagens de *Graph RAG*, para conectar documentos, entidades e relações, quanto em sistemas de memória de agente, para representar relações entre experiências, conceitos, tarefas ou preferências. Porém, o uso de técnicas semelhantes não significa que os dois conceitos sejam iguais (HU *et al.*, 2026).

A principal diferença está no papel que essas informações possuem no sistema. Em sua forma mais comum, o RAG consulta uma base externa para complementar uma resposta em uma tarefa específica. Já a memória de agente está relacionada ao

registro, organização, atualização e recuperação de informações geradas ao longo das próprias interações do agente. Dessa forma, enquanto o RAG funciona principalmente como um mecanismo de acesso a conhecimento externo, a memória de agente atua como uma estrutura persistente, capaz de evoluir com a experiência do agente (HU *et al.*, 2026; DU, P., 2026).

2.2.3 Engenharia de Contexto

A engenharia de contexto (*Context Engineering*) está relacionada à forma como as informações são organizadas, selecionadas e inseridas na janela de contexto do LLM. Em vez de tratar o contexto apenas como um espaço onde todo o histórico pode ser acumulado, essa abordagem considera a janela de contexto como um recurso limitado, que precisa ser gerenciado de forma cuidadosa. Isso inclui decidir quais instruções, mensagens anteriores, documentos recuperados, respostas de ferramentas e memórias devem ser enviados ao modelo em cada chamada (HU *et al.*, 2026).

Essa abordagem possui uma relação direta com a memória de agente, pois muitas técnicas utilizadas para manter uma boa memória também dependem de um bom gerenciamento de contexto. Em interações longas, não basta salvar todas as informações disponíveis e reenviá-las ao modelo, pois isso aumenta o custo, aproxima o sistema do limite de tokens e pode prejudicar a qualidade da resposta. Por isso, mecanismos como seleção de informações relevantes, sumarização, compactação e organização do histórico são importantes tanto para a engenharia de contexto quanto para a memória de agente (LIU *et al.*, 2023; DU, Y. *et al.*, 2025). Os dois caminham juntos, mas não são o mesmo conceito. A engenharia de contexto trata de organizar o que entra no modelo em uma chamada específica. A memória de agente, por sua vez, abrange não só essa organização pontual, mas também a persistência, atualização e recuperação de informações ao longo do tempo e entre sessões.

No presente trabalho, a engenharia de contexto será uma estratégia importante para o funcionamento do agente. O sistema de memória não deve apenas armazenar informações, mas também ajudar a escolher o que deve entrar ou não no contexto em cada interação. Por isso, a solução proposta precisa cuidar constantemente dos limites de tokens e da quantidade de contexto enviada ao LLM, mantendo apenas as informações necessárias para a tarefa atual. Dessa forma, a memória de agente define o que o sistema consegue lembrar entre interações, decidindo o que deve ser preservado e recuperado posteriormente pelo agente, enquanto a engenharia de contexto define como parte dessas informações será apresentada ao modelo no momento da resposta.

2.3 FORMAS DE REPRESENTAÇÃO DE MEMÓRIA

Existem diferentes formas pelas quais a memória pode ser representada em um agente. Segundo Hu *et al.* (2026), uma forma comum de organizar essas representações é separar a memória em três categorias: nível de token, paramétrica e latente. No contexto deste trabalho, o foco está na memória em nível de token, pois ela representa informações, externas ao modelo, que podem ser armazenadas, consultadas, atualizadas e removidas ao longo do tempo. As outras duas representações estão relacionadas aos parâmetros ou aos estados internos do LLM.

Dentro da memória em nível de token, existem diferentes formas de organizar as unidades armazenadas. Nas subseções seguintes, serão tratados principalmente os tipos *Flat* e Planar, que são mais próximos das estratégias consideradas para guardar informações do agente. A representação hierárquica também faz parte dessa taxonomia, mas será apresentada apenas de forma breve, pois não é o foco deste trabalho.

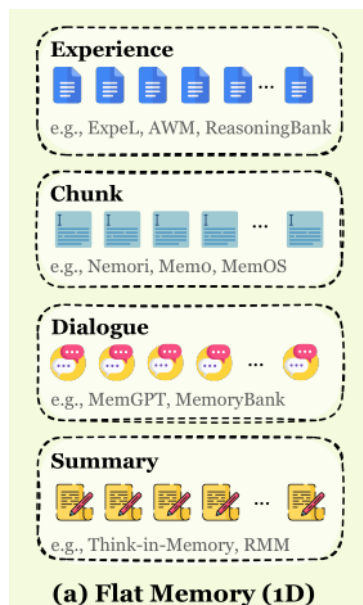
2.3.1 Flat

A representação plana (*Flat Memory*) armazena informações como unidades discretas, sem modelar explicitamente as relações entre elas. Essas unidades podem ser trechos de texto, registros de conversa, perfis de usuário, experiências anteriores ou representações vetoriais associadas a essas informações. Nesse tipo de memória, cada item pode ser armazenado e recuperado individualmente, mas a relação entre diferentes itens não está representada diretamente na estrutura da memória, como ilustrado na Figura 6 (HU *et al.*, 2026).

A grande vantagem é dada devido a simplicidade de implementação e a facilidade de crescimento. Novas memórias podem ser adicionadas, removidas ou recuperadas por busca semântica sem a necessidade de manter uma estrutura complexa entre os registros. Por isso, ela é adequada para sistemas que precisam acumular informações de forma incremental, como históricos de interação, preferências de usuários ou registros de experiências do agente.

Em contrapartida, o limite dessa abordagem é que a relação entre as memórias depende principalmente do mecanismo de recuperação. Conforme a quantidade de registros aumenta, podem surgir redundâncias, ruídos e memórias pouco relevantes. Além disso, como não há uma estrutura explícita entre os itens, o sistema pode recuperar informações relacionadas sem compreender claramente como elas se conectam entre si (HU *et al.*, 2026).

Figura 6 – Representação plana de memória.



Fonte: Hu *et al.* (2026).

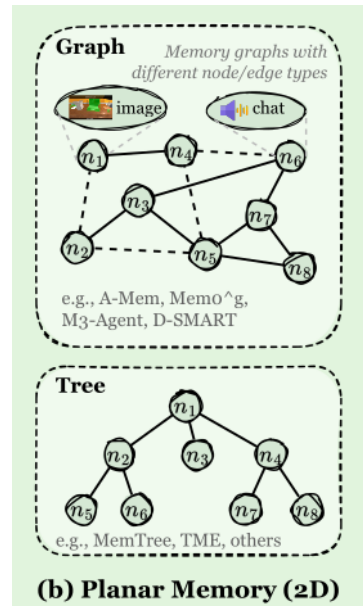
2.3.2 Planar

A representação planar (*Planar Memory*) adiciona uma estrutura explícita entre as unidades de memória, mas ainda dentro de uma única camada, aproximando-se de um modelo relacional entre registros. Essa estrutura pode ser representada por grafos, árvores, tabelas ou outras formas de conexão entre registros. Diferente da memória plana, a memória planar não trata cada item apenas como um registro isolado, mas também considera as relações existentes entre eles, como ilustrado na Figura 7 (HU *et al.*, 2026).

Essa forma de representação é útil quando o sistema precisa armazenar relações entre conceitos, eventos, tarefas, usuários ou experiências anteriores. Em uma estrutura baseada em grafos, por exemplo, cada memória pode ser representada como um nó, enquanto as relações entre memórias podem ser representadas como arestas. Isso permite que o agente não apenas recupere uma informação específica, mas também navegue por informações relacionadas.

A representação planar apresenta custos elevados de construção e de busca, o que pode dificultar sua aplicação em sistemas que crescem em volume e diversidade de informações. Em compensação, sua estrutura habilita o raciocínio sobre relações, permitindo que o agente combine múltiplas memórias e percorra as conexões entre elas para apoiar consultas mais complexas (HU *et al.*, 2026).

Figura 7 – Representação planar de memória.



Fonte: Hu *et al.* (2026).

2.3.3 Hierárquica

Por fim, a representação hierárquica (*Hierarchical Memory*) organiza as memórias em múltiplos níveis de abstração. Nesse formato, informações mais específicas, como registros brutos de interação, podem ser conectadas a resumos, categorias ou conceitos mais gerais. Dessa forma, o agente pode navegar tanto por memórias detalhadas quanto por representações mais compactas e abstratas (HU *et al.*, 2026).

Apesar de ser uma forma poderosa de organização, a memória hierárquica exige mecanismos mais complexos de construção, manutenção e atualização. Por esse motivo, ela não é tratada como foco principal deste trabalho, mas ajuda a situar possibilidades futuras para sistemas de memória mais sofisticados.

2.4 FUNÇÕES DE MEMÓRIA EM AGENTES

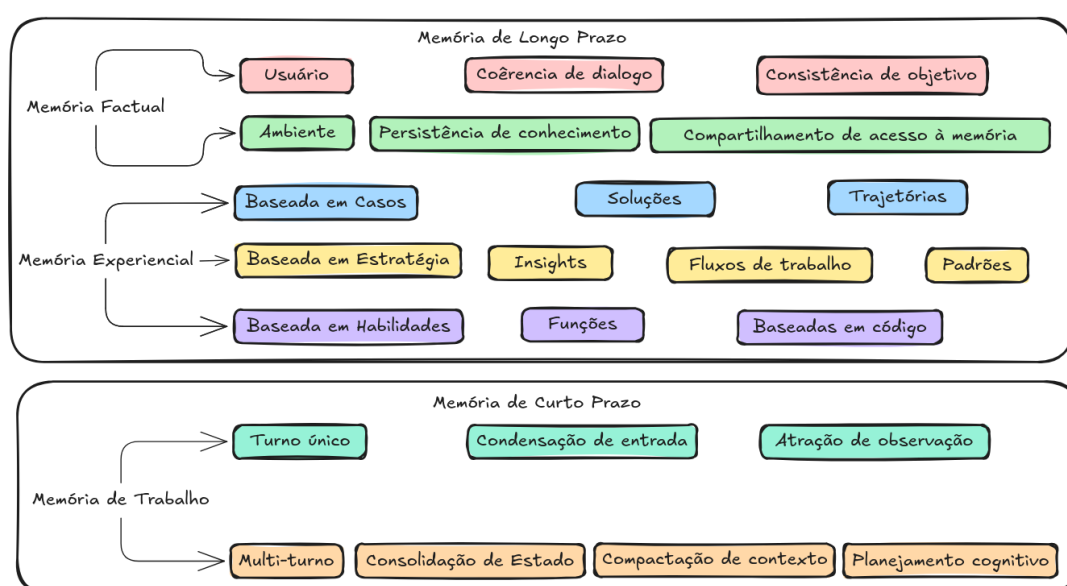
Nesta seção, o foco será qual função a memória exerce no comportamento do agente. Assim, busca-se compreender por que a memória é necessária e que tipo de informação ela permite ao agente manter, recuperar e utilizar durante suas interações.

Como mencionado anteriormente, a memória pode ser tanto de curto prazo, apoiando a interação atual, quanto de longo prazo, preservando informações entre sessões. Entretanto, além dessas diferenças, a literatura também organiza a memória de acordo com sua função que ela desempenha na arquitetura do sistema, incluindo o armazenamento de fatos, experiências anteriores e informações temporárias utilizadas na execução de uma tarefa (ZHANG, Z. *et al.*, 2024). Essas funções são frequen-

temente relacionadas em analogias com a memória humana, com a capacidade de manter fatos, acumular experiências e manipular informações relevantes durante uma tarefa.

Neste trabalho, será adotada a taxonomia proposta por Hu *et al.* (2026) que apresentam três funções principais de memória em agentes: factual, experiencial e de trabalho como mostrado na Figura 8. Essa divisão ajuda a diferenciar memórias usadas para manter conhecimento persistente daquelas utilizadas para apoiar o raciocínio e tomada de decisão em uma interação específica.

Figura 8 – Três funções de memória em agentes de IA.



Fonte: Adaptado de Hu *et al.* (2026).

2.4.1 Factual

A memória factual (*Factual Memory*) refere-se à capacidade do agente de armazenar e recuperar fatos sobre eventos passados, informações do usuário e o estado do ambiente. Esses fatos são informações que posteriormente o agente pode consultar e dividir pela origem de cada informação, como por exemplo saber que o usuário trabalha em determinado setor ou que o usuário prefere respostas curtas. Esses conhecimentos se encaixam em duas categorias da psicologia cognitiva: a memória episódica, que registra eventos vividos pelo agente, como interações passadas, e a memória semântica, que armazena fatos e conhecimentos gerais sobre o mundo (TULVING, 1972 apud HU *et al.*, 2026).

Segundo Pengfei Du (2026), essa arquitetura precisa garantir três propriedades fundamentais, sendo elas:

- **Consistência:** Preserva um estado interno persistente em relação aos fatos.
- **Coerência:** Lembra de informações importantes e conseguir conectar memórias diferentes garantindo um dialogo conectado ao invés de informações isoladas
- **Adaptabilidade:** Capaz de personalizar a experiência do usuário com base em perfis e feedbacks, levando a tomada de decisões e estilos de resposta que agradem ao usuário.

A memória factual pode ser dividida em duas categorias: Memória Factual do Usuário (*User Factual Memory*) e Memória Factual do Ambiente (*Environment Factual Memory*).

2.4.1.1 Memória Factual do Usuário

A memória factual do usuário persiste fatos sobre um usuário específico entre sessões e tarefas, incluindo preferências, padrões de uso e informações estáveis da interação (HU *et al.*, 2026). Esses fatos são construídos a partir de padrões observados ao longo do tempo, ou pedidos que o próprio usuário faz em uma interação e permitem que o agente ajuste seu comportamento sem que o usuário precise repetir as mesmas orientações a cada nova sessão.

No contexto da LivIA, agente desenvolvido neste trabalho, um usuário pode preferir visualizar os indicadores de Eficácia Geral do Equipamento (*Overall Equipment Effectiveness*) (OEE) dos centros de trabalho em formato de tabela. Ao identificar esse padrão e registrá-lo como fato, o agente passa a adotar esse formato nas respostas subsequentes, sem necessidade de nova instrução explícita. Dessa forma, a memória factual do usuário sustenta a continuidade e a personalização da interação, tornando o comportamento do agente progressivamente mais alinhado às preferências individuais.

2.4.1.2 Memória Factual do Ambiente

A memória factual do ambiente compreende fatos sobre entidades e estados externos ao usuário, abrangendo documentos, bases de código, ferramentas e registros de interação com o ambiente (HU *et al.*, 2026). Esses fatos dizem respeito ao mundo externo em que as tarefas são executadas, formando uma camada de referência persistente e recuperável ao longo das sessões.

No contexto da LivIA, a memória factual do ambiente abrange informações sobre o sistema produtivo, como quais produtos podem ser fabricados em determinados centros de trabalho ou quando ocorrem paradas planejadas. Ao registrar esses fatos, o agente passa a considerá-los ao interpretar consultas do usuário, sem precisar solicitá-los novamente a cada interação. Dessa forma, a memória factual do ambiente

permite que o agente mantenha um entendimento consistente do contexto operacional, apoiando respostas mais precisas e contextualmente fundamentadas.

2.4.2 Trabalho

A memória de trabalho (*Working Memory*) refere-se ao conjunto de ferramentas responsáveis pela gestão e manipulação do contexto durante a execução de uma tarefa ou sessão (HU *et al.*, 2026). Como discutido na Seção 2.1.3, a janela de contexto do LLM funciona como um canal de entrada, que recebe os dados brutos até atingir seu limite, sem capacidade própria de priorizar as informações mais relevantes. A memória de trabalho atua sobre essa limitação, transformando a janela de contexto em um espaço gerenciável, no qual o conteúdo pode ser filtrado, comprimido e reorganizado para preservar a coerência do raciocínio do agente.

Essa gestão pode ser tratada em duas ocasiões, conforme a interação: turno único (*single-turn*) e multi-turno (*multi-turn*) (HU *et al.*, 2026). O presente projeto concentra-se no cenário multi-turno, por se tratar de um *chatbot* onde o gerenciamento do contexto acumulado ao longo dos turnos se torna crítico para a continuidade do agente, assim evitando alucinações.

2.4.2.1 Turno único

A memória de trabalho em turno único trata-se do processamento de grandes entradas em uma única requisição ao modelo, como documentos longos ou grandes volumes de dados recebidos de uma só vez. Nesse cenário, em vez de enviar o conteúdo bruto integralmente, aplicam-se técnicas de condensação de entrada, que reduzem a quantidade de tokens preservando as informações essenciais para a tarefa (HU *et al.*, 2026).

Esse tipo de gerenciamento está diretamente relacionado à engenharia de contexto, discutida na Seção 2.2.3, uma vez que ambos tratam do uso eficiente da janela de contexto como um recurso limitado.

2.4.2.2 Multi-turno

A memória de trabalho multi-turno trata da manutenção do estado ao longo das interações com o agente. Neste contexto, define-se turno como cada par de mensagem e resposta trocado entre o usuário e o agente, e sessão como o conjunto de turnos realizados em uma mesma janela de interação contínua. O problema surge do acúmulo progressivo do histórico a cada novo turno. Mesmo com janelas de contexto amplas, com novos modelos tendo até um milhão de contexto, esse acúmulo divide a atenção do modelo, eleva a latência e pode levar ao desvio do objetivo da tarefa (*goal drift*) (HU *et al.*, 2026).

Para lidar com isso, a memória de trabalho multi-turno atua como um portador de estado externalizado, organizando um ciclo contínuo de leitura, avaliação e escrita das informações relevantes (HU *et al.*, 2026). Em vez de reenviar todo o histórico bruto a cada turno, o sistema mantém um estado compacto, atualizado de forma incremental, que preserva o que é essencial à continuidade da tarefa e descarta redundâncias. Nesse sentido, a memória de trabalho multi-turno vai além da engenharia de contexto: enquanto esta decide o que entra no *prompt* em uma chamada isolada, a memória de trabalho opera como uma arquitetura contínua de gerenciamento de estado ao longo de toda a sessão, compactando e reorganizando o histórico a cada turno para que o agente mantenha coerência do início ao fim da interação.

Entre as estratégias adotadas para esse fim, destaca-se a consolidação de estado (*state consolidation*), que mapeia uma trajetória de interações em crescimento para um espaço de estado de tamanho fixo (HU *et al.*, 2026). O MEM1, por exemplo, mantém um estado interno compartilhado que integra as novas observações com a memória anterior, descartando informações irrelevantes ou redundantes a cada turno (ZHOU *et al.*, 2025).

2.4.3 Experiencial

A memória experiencial (*Experiential Memory*) refere-se à capacidade do agente de acumular trajetórias, estratégias e resultados de interações anteriores, aprendendo com acertos e erros para aprimorar progressivamente seu desempenho na resolução de tarefas (HU *et al.*, 2026). Esse aprendizado é armazenado de forma explícita, permitindo que o agente ajuste seu comportamento sem a necessidade de atualizar os pesos do modelo de linguagem. Por depender de mecanismos de reflexão e abstração mais complexos, a memória experiencial não integra o foco deste trabalho, sendo tratada como possibilidade de trabalho futuro. Assim, o desenvolvimento realizado concentra-se nas memórias factual e de trabalho.

2.5 DINÂMICAS DO CICLO DE VIDA DA MEMÓRIA

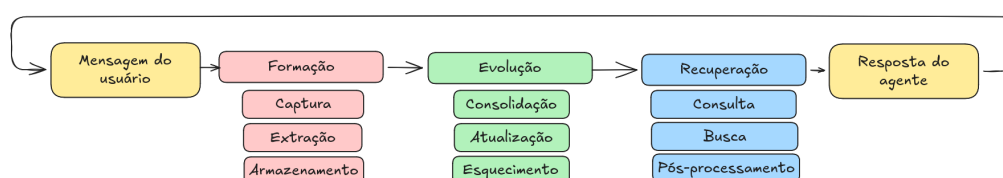
A memória do agente possui um ciclo de vida que define como uma informação nasce, se modifica e é trazida de volta quando relevante. Esse ciclo é constituído por três processos:

- **Formação:** O conteúdo bruto das interações é convertido em unidades de conhecimento.
- **Evolução:** As unidades de conhecimento são consolidadas com a base já registrada ou atualizadas quando há informações novas ou conflitantes.

- **Recuperação:** A informação certa é localizada e disponibilizada para apoiar a próxima ação do agente.

Esses processos não atuam de forma isolada. Eles são continuamente acionados ao longo do uso, como um *loop* de *feedback*, em que cada interação alimenta a formação, que aciona a evolução e prepara a base para uma recuperação mais precisa nas próximas interações, conforme ilustrado na Figura 9. É essa sequência de passos que torna a memória uma estrutura adaptativa, em que o agente não apenas armazena, mas também reorganiza e reutiliza o que sabe, sustentando coerência e consistência ao longo do tempo (HU *et al.*, 2026).

Figura 9 – Dinâmicas do ciclo de vida da memória do agente.



Fonte: Adaptado de Hu *et al.* (2026).

2.5.1 Formação

A formação é o processo pelo qual a memória do agente surge. A cada interação, como: diálogos com usuários, observações do ambiente ou retornos de ferramentas, o agente recebe informações em sua forma bruta, ou *verbatim*. A partir desse conteúdo, o sistema identifica o que é relevante e o transforma em unidades compactas de conhecimento, que podem apoiar tanto a interação atual quanto a serem preservadas para uso posterior. Diferente de uma simples gravação literal, essa etapa envolve uma seleção ativa do que vale a pena reter, conforme o tipo de informação e o uso pretendido (HU *et al.*, 2026).

2.5.1.1 Sumarização Semântica

A sumarização semântica transforma o conteúdo bruto das interações em resumos compactos que preservam o significado principal da informação, em vez de manter cada detalhe específico. O objetivo é reter a essência, como o tema central de uma conversa, o fluxo de uma tarefa em execução ou o perfil acumulado de um usuário, descartando redundâncias e elementos que não contribuem para o raciocínio futuro (HU *et al.*, 2026). Por exemplo, após uma sessão em que o usuário e o agente discutiram preferências de comunicação, a sumarização produziria algo como: “O usuário prefere respostas diretas e sem formatação excessiva.”

Essa estratégia responde a três limitações centrais do envio contínuo do histórico completo ao modelo, identificadas por Zhou *et al.* (2025). A primeira é o crescimento do custo de inferência e do uso de memória. As arquiteturas baseadas em *Transformers* apresentam custo computacional quadrático em relação ao número de tokens, tornando o processamento de históricos longos cada vez mais custoso. A segunda é a queda de desempenho conforme o volume do histórico cresce. Ainda que modelos recentes apresentem janelas de contexto cada vez maiores, há um ponto a partir do qual o aumento do número de tokens deteriora o raciocínio, seja por ultrapassar o intervalo em que o modelo foi treinado a operar com eficácia, seja por dificuldades próprias do processamento de entradas muito longas. A terceira é a diluição da atenção. Mesmo quando o histórico cabe na janela disponível, o acúmulo de informações irrelevantes ou redundantes reduz a capacidade do modelo de focar no que é relevante para a resposta.

2.5.1.2 Destilação de Conhecimento

A destilação de conhecimento extrai elementos específicos das interações que possam ser reutilizados em momentos futuros. O conteúdo destilado pode incluir fatos sobre o usuário, intenções, preferências, estratégias bem-sucedidas ou outras informações úteis para o agente (HU *et al.*, 2026). Por exemplo, da mesma sessão, a destilação extrairia fatos estruturados como `prefere_respostas: diretas e evitar: listas longas`, que podem ser recuperados individualmente em interações futuras sem precisar do resumo completo.

A motivação para essa estratégia parte do problema observado por Lewis (2026): o histórico de uma conversa carrega informação útil para o agente, mas mantê-lo em sua forma literal passa a atrapalhar o desempenho à medida que cresce.

Em sua proposta, cada interação é representada como um objeto composto por alguns campos que capturam o essencial da troca, como o conteúdo principal do diálogo e seu contexto. Esse tipo de estruturação permite que informações úteis ao longo do tempo, como preferências e padrões recorrentes, sejam mantidas e recuperadas sem a necessidade de consultar o histórico literal.

2.5.1.3 Construção Estruturada

A construção estruturada organiza o conteúdo das interações em estruturas conectadas, como grafos de conhecimento ou árvores hierárquicas. Em vez de armazenar a informação apenas como texto bruto, as entidades e relações presentes nas interações são extraídas e conectadas, formando uma estrutura que pode ser percorrida posteriormente para apoiar consultas mais complexas (HU *et al.*, 2026).

Esse tipo de organização favorece a transparência da memória, já que cada relação é representada de forma explícita, e habilita o raciocínio em múltiplos passos

(*multi-hop*), em que a resposta a uma consulta depende da combinação de múltiplas entidades conectadas. Em contrapartida, exige mecanismos de extração de entidades e relações na etapa de formação, o que aumenta a complexidade do processo.

2.5.1.4 Representação Latente

A representação latente codifica o conteúdo das interações diretamente em formatos internos ao modelo, como vetores de *embedding* ou estados de *key-value cache*, mantidos em um espaço latente contínuo (HU *et al.*, 2026). Por operar sobre representações internas do modelo, em vez de unidades de texto explícitas, essa abordagem não se enquadra na memória em nível de token.

2.5.1.5 Internalização Paramétrica

A internalização paramétrica consolida memórias diretamente no espaço de pesos do modelo, por meio de atualizações de seus parâmetros, como o *fine-tuning*. Nesse mecanismo, a informação deixa de ser armazenada externamente e passa a integrar o comportamento aprendido pelo modelo (HU *et al.*, 2026). Por modificar o próprio modelo em vez de manter o conhecimento como unidades explícitas de texto.

2.5.2 Evolução

A evolução trata da integração das novas unidades de conhecimento à base já registrada pelo agente. Apenas acumular cada memória ao final do repositório acaba gerando inconsistências, redundâncias e perda de validade conforme o tempo passa, motivando uma etapa dedicada a reorganizar e manter o conteúdo armazenado de forma consistente, compacta e relevante ao longo do uso (HU *et al.*, 2026).

2.5.2.1 Consolidação

A consolidação cruza as memórias novas com o que já está registrado, juntando as que se relacionam em representações mais gerais. Em vez de manter cada memória como um registro isolado, o processo extrai padrões e generalizações que podem ser reutilizados em situações similares, acumulando aprendizado ao longo do tempo (HU *et al.*, 2026). Sistemas como o RMM aplicam essa ideia recuperando, para cada nova memória, um conjunto de candidatas similares já registradas, e deixando que o modelo de linguagem decida se a fusão é apropriada (TAN *et al.*, 2025).

Como esse processo se baseia em abstração, existe um risco. Detalhes específicos podem ser perdidos ao longo do tempo, fazendo com que o agente perca sensibilidade a eventos raros ou exceções. O quanto vale generalizar e o quanto vale preservar essas especificidades depende do tipo de tarefa que o agente apoia (HU *et al.*, 2026).

2.5.2.2 Atualização

A atualização revisa ou substitui memórias quando o agente recebe informações novas que contradizem o que já está registrado. É uma operação pontual sobre entradas específicas, como corrigir um fato, substituir uma preferência ou ajustar algo que perdeu validade (HU *et al.*, 2026). Por exemplo, uma preferência antes registrada como “idioma português” precisa dar lugar à nova preferência “idioma inglês” caso o usuário sinalize a mudança.

O desafio aqui é decidir quando uma informação nova deve sobrescrever a antiga e quando ela deve ser tratada como ruído. Substituir cedo demais pode descartar conhecimento ainda válido, e manter algo obsoleto compromete o raciocínio do agente. Essa decisão envolve um equilíbrio entre estabilidade e adaptação (HU *et al.*, 2026).

2.5.2.3 Esquecimento

O esquecimento é o descarte intencional de informações desatualizadas, redundantes ou de baixo valor, com o objetivo de manter a base focada no que ainda é relevante. Sem esse mecanismo, a base cresce sem limite, prejudicando a recuperação e gerando ruído nas decisões do agente (HU *et al.*, 2026).

O critério para descartar uma memória pode considerar o tempo desde que ela foi criada, a frequência com que foi acessada ou uma combinação desses sinais com a sua relevância (HU *et al.*, 2026). Zhong *et al.* (2024), por exemplo, aplicam à memória de agente a curva de esquecimento de Ebbinghaus, um conceito da psicologia cognitiva que descreve como uma informação se perde ao longo do tempo quando ela não é reforçada por nova utilização. Esse mecanismo imita o comportamento humano de esquecer o que não é revisitado e reforçar o que é acessado com frequência.

2.5.3 Recuperação

A recuperação é a etapa em que o conhecimento armazenado pelo agente volta a ser utilizado. Mesmo com uma base bem formada e bem mantida, esse conhecimento só tem efeito quando a memória certa chega ao modelo no momento certo, antes de o agente gerar a resposta. Por isso, a recuperação envolve várias decisões: identificar quando consultar a memória, como formular a consulta, qual estratégia de busca utilizar e como tratar o que foi recuperado antes de incorporá-lo ao contexto (HU *et al.*, 2026).

2.5.3.1 Tempo e Intenção

O primeiro passo da recuperação é decidir quando consultar a memória e de qual fonte buscar a informação. Algumas abordagens fazem a busca sempre, a cada interação, para garantir que o conteúdo relevante chegue ao contexto. Outras deixam

a decisão a cargo do próprio modelo, que avalia se a consulta atual precisa ou não de informações da memória (HU *et al.*, 2026). O MemGPT, por exemplo, expõe a busca como uma ferramenta que o modelo pode chamar quando julgar necessário, dentro de uma arquitetura inspirada em sistemas operacionais (PACKER *et al.*, 2023). Cada estratégia tem seu compromisso, buscar sempre evita esquecer mas adiciona ruído e custo a cada turno. Enquanto deixar o modelo decidir reduz esse *overhead* mas corre o risco de o agente confiar demais no que acredita saber e gerar respostas sem base no que foi de fato registrado.

2.5.3.2 Construção da Consulta

Depois de decidido que vale buscar, é preciso transformar a entrada do usuário em uma consulta efetiva para a memória. A pergunta original nem sempre coincide com a forma como a informação foi armazenada, então o sistema pode reescrever a consulta para se aproximar do índice da memória, ou decompor uma pergunta complexa em sub-consultas menores, recuperando cada parte separadamente e combinando os resultados (HU *et al.*, 2026). A qualidade dessa formulação influencia diretamente o que será encontrado, e por isso vem recebendo atenção crescente da literatura como uma etapa em si, e não como um passo trivial antes da busca.

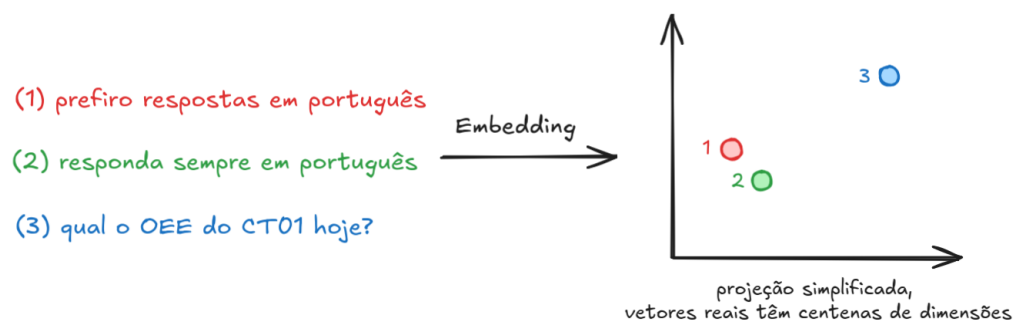
2.5.3.3 Estratégias de Busca

Com a consulta pronta, a busca executa o acesso à memória. As estratégias variam conforme o tipo de correspondência que se busca. A busca lexical compara a consulta diretamente com o conteúdo armazenado a partir das palavras presentes em ambos, sendo rápida e interpretável, mas pouco tolerante a variações de linguagem. A busca semântica, hoje a forma mais comum em sistemas de memória de agente, codifica consulta e memória em vetores no mesmo espaço e mede a proximidade entre eles, capturando relações de significado mesmo quando os termos diferem (HU *et al.*, 2026). Sistemas como o MemoryBank (ZHONG *et al.*, 2024) e o RMM (TAN *et al.*, 2025) se apoiam nessa abordagem para recuperar memórias relacionadas ao tema atual. Quando a memória é organizada em grafo, a busca pode percorrer ligações entre entidades, suportando consultas que dependem de combinar múltiplas memórias. Abordagens híbridas combinam esses paradigmas, como em Generative Agents, em que a relevância é calculada por uma combinação de similaridade semântica, recência e importância (PARK *et al.*, 2023).

Na busca semântica, o texto é convertido em um vetor de números reais por um modelo de *embedding*. Cada vetor representa um ponto num espaço de alta dimensão, onde textos com significados próximos ocupam regiões próximas, independente das palavras usadas. A medida de proximidade adotada é a distância cosseno, que compara a direção dos vetores, não o quão longe eles estão. Essa distinção é importante

porque a direção carrega o significado do texto, enquanto o comprimento do vetor reflete características irrelevantes para a semântica, como o tamanho da entrada. Assim, “seja conciso nas respostas” e “responda de forma breve e direta” apontam para direções próximas no espaço de *embedding*, mesmo sem compartilhar palavras, e a distância cosseno os aproxima corretamente. A Figura 10 ilustra esse comportamento com três frases de exemplo.

Figura 10 – Projeção simplificada de três frases no espaço de *embedding*



Fonte: Elaborado pelo autor.

2.5.3.4 Pós-Processamento

Por último, o conteúdo recuperado nem sempre vai direto para o modelo. Antes da geração da resposta, os fragmentos retornados podem ser reorganizados ou filtrados, descartando memórias com baixa relevância e priorizando as que mais se conectam à consulta. Também é comum agregar e comprimir os trechos recuperados, juntando informações próximas e reduzindo redundâncias antes de incluir tudo no contexto (HU *et al.*, 2026). Esse pós-processamento é o que evita que o agente receba um conjunto longo, ruidoso e potencialmente contraditório de informações, mantendo o contexto final compacto e coerente.

A fundamentação apresentada neste capítulo estabelece as bases sobre as quais o restante do trabalho se apoia. Com essas noções definidas, o Capítulo 3 detalha o problema tratado no projeto e os requisitos técnicos derivados dele.

3 DESCRIÇÃO DO PROBLEMA E REQUISITOS TÉCNICOS

Este capítulo apresenta o contexto, o problema e os requisitos que guiam o projeto.

A Seção 3.1 descreve o cenário em que o agente opera e o que motivou a proposta. A Seção 3.2 define o problema com precisão técnica. Por fim, a Seção 3.3 lista os requisitos funcionais e não-funcionais que orientam as decisões de projeto.

3.1 CONTEXTO E MOTIVAÇÃO

O LiveMES é um sistema Sistema de Execução da Manufatura (*Manufacturing Execution System*) (MES) que apresenta, em tempo real, o que está acontecendo no chão de fábrica. Ele é utilizado por operadores industriais que monitoram centros de trabalho em turnos. Cada operador acompanha um conjunto específico de máquinas, linhas e indicadores. Ao longo do tempo, cada um acumula um conhecimento detalhado sobre o ambiente em que trabalha, por exemplo, quais os problemas mais frequentes no seu setor, o que costuma ser feito para resolvê-los e quais informações do turno anterior são necessárias para executar bem o turno atual.

Para atender a essa demanda, foi desenvolvida a LivIA, um agente de IA integrado ao LiveMES que permite aos operadores consultar informações do chão de fábrica por linguagem natural. Em vez de navegar por diferentes telas ou depender da memória do operador para lembrar o que aconteceu nos turnos anteriores, o operador pode perguntar diretamente sobre qualquer dado de produção existente no sistema. O agente pode então raciocinar sobre esses dados, identificando padrões, sugerindo causas para um indicador abaixo do esperado ou trazendo informações que não seriam óbvias na leitura direta dos dados. Nesse fluxo, o agente precisa compreender não apenas a pergunta, mas também quem está perguntando e qual seu contexto.

Com o uso contínuo, certas informações se tornam relevantes a cada sessão, como o setor de atuação, o turno que cobre e as preferências de formato de resposta. Sem memória, o agente começa cada conversa do zero e o operador precisa repetir informações já fornecidas antes para obter respostas adequadas ao seu perfil. Enquanto isso não acontece, o agente responde de forma genérica, sem adaptar as respostas a quem está perguntando.

A LivIA opera como agente central da plataforma, servindo operadores de múltiplas empresas simultaneamente. Por ter acesso a dados de diferentes organizações, o agente precisa garantir, a cada consulta, que a resposta gerada corresponda exclusivamente ao usuário e à empresa corretos. As memórias persistidas seguem o mesmo princípio de que cada fato armazenado é indexado por identificador de usuário e de empresa, de forma que uma recuperação nunca retorne registros de outro usuário ou outra organização. Se esse isolamento falhar, o agente pode sugerir dados pertencen-

tes a outra empresa, expondo informações operacionais a usuários não autorizados e infringindo a Lei Geral de Proteção de Dados.

Em um ambiente industrial, essa perda de continuidade tem consequências diretas. Sem o histórico adequado, o agente pode alucinar respostas plausíveis mas incorretas, atribuindo valores, ocorrências ou estados a equipamentos que nunca foram informados na sessão atual. Quando a resposta do agente informa uma decisão operacional, o custo de um erro cresce. Uma parada de linha, uma alocação incorreta de recursos ou um diagnóstico equivocado de falha podem ter impacto direto na produção. Nesses casos, a falta de consistência entre sessões não é apenas um inconveniente, mas um risco real para a operação.

Esse desafio não é exclusivo da LivIA. Com o crescimento do uso de agentes baseados em Modelo de Linguagem de Grande Escala (*Large Language Model*) (LLM), o problema já motivou projetos dedicados. O Hermes Agent (NOUS RESEARCH, 2026), da Nous Research, acumula mais de 169 mil estrelas no GitHub tratando memória como funcionalidade central. O MemGPT (PACKER *et al.*, 2023), desenvolvido na UC Berkeley, parte da mesma premissa de que sem memória persistente, agentes de uso contínuo perdem contexto, repetibilidade e utilidade. Porém, ter memória não é suficiente. A forma como ela é implementada define se o agente ganha em consistência ou passa a falhar de maneiras novas. A Seção 3.2 detalha por que esse problema é desafiador mesmo depois de adicionada a memória ao agente.

3.2 DEFINIÇÃO DO PROBLEMA

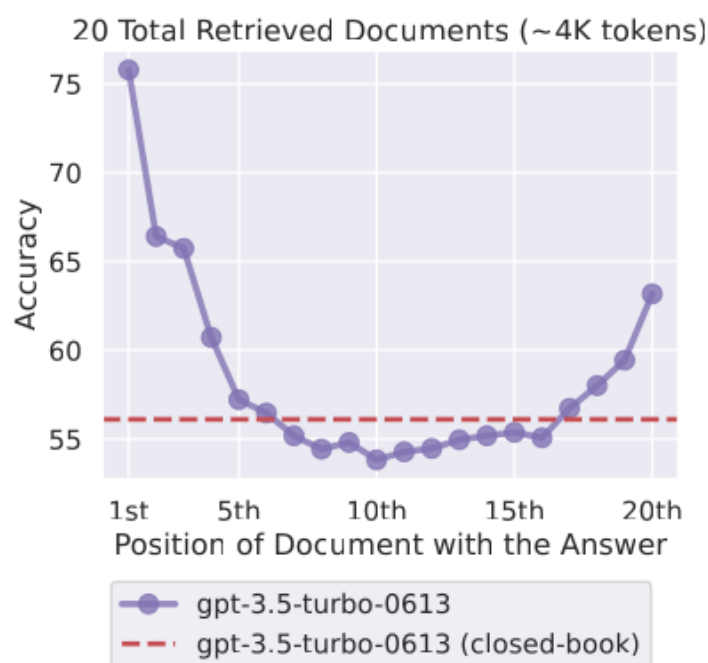
O problema central está na natureza dos modelos de linguagem. LLMs são *stateless* por design, então cada chamada à API é independente das anteriores, e o modelo não mantém nenhum estado entre requisições (DU, P., 2026). E isso não é um erro arquitetural, elas foram projetadas para serem assim porque sem estado interno, os modelos podem rodar em paralelo e escalar sem estado compartilhado, atendendo a milhares de usuários simultaneamente. Para um agente de uso contínuo como a LivIA, a consequência é direta, pois toda informação relevante precisa ser diretamente fornecida a cada nova chamada. O histórico de interações anteriores, as preferências expressas pelo operador e os fatos sobre o ambiente operacional não persistem entre sessões. Portanto, se não forem reenviados, simplesmente não existem para o modelo.

Uma alternativa direta seria acumular o histórico completo no contexto e depois enviar ao modelo a cada turno. Porém, essa estratégia esbarra em limites práticos. O texto de entrada é processado em tokens, e cada modelo possui uma janela de contexto com um custo computacional associado. As APIs dos principais modelos comerciais cobram por token processado, e contextos extensos aumentam diretamente o custo operacional por consulta (ANTHROPIC, 2026). Quanto maior o histórico enviado, maior também a latência para geração da resposta. Cada modelo também possui um teto

físico para o tamanho do contexto, como o Claude Opus, com suporte a até 1 milhão de tokens (ANTHROPIC, 2026). Quando esse limite é atingido, mensagens antigas precisam ser descartadas.

Mais importante do que o custo é a limitação de qualidade. Liu *et al.* (2023) documentam um fenômeno conhecido como *Lost in the Middle*, onde os modelos de linguagem utilizam melhor as informações posicionadas no início do contexto, com desempenho inferior quando a informação relevante está no meio. Os experimentos foram feitos através de perguntas para o LLM, onde a resposta estava contida em meio a múltiplos documentos. Quando o documento relevante é deslocado do início para posições centrais do contexto, foi observado que a acurácia cai entre 20 e 30 pontos percentuais em relação ao desempenho. Os autores observam esse comportamento como uma curva em U assimétrica, conforme ilustrado na Figura 11, onde o desempenho é mais alto no início, cai progressivamente em direção ao meio e se recupera parcialmente no final do contexto, consistente em diferentes modelos testados. Em um agente que acumula histórico ao longo de sessões longas, isso significa que informações relevantes podem estar presentes no contexto e ainda assim ser ignoradas, mal aproveitadas ou substituídas por informações menos relevantes que ocupam posições mais favoráveis.

Figura 11 – Desempenho em função da posição da informação relevante no contexto.



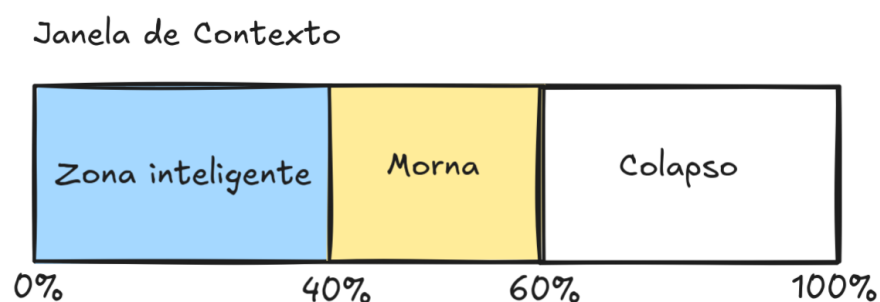
Fonte: Liu *et al.* (2023)

O problema é reforçado por Yufeng Du *et al.* (2025), que buscavam saber se a degradação de desempenho em contextos longos era causada pela distração, com informações irrelevantes no meio, ou pelo comprimento em si. Com esse objetivo, ga-

rantiram em seus experimentos que o documento relevante estava sempre presente no contexto, isolando o efeito do comprimento da entrada e avaliando tarefas de matemática, resposta a perguntas e programação. Mesmo assim, o desempenho caiu de forma significativa conforme o contexto cresceu. O achado mais relevante foi que a degradação persistiu mesmo quando o contexto foi reduzido apenas às evidências relevantes, sem conteúdo adicional ao redor. Portanto, a causa não é a distração, mas o próprio comprimento do contexto, o que mostra que não basta a informação estar presente e bem posicionada.

Hong, Troynikov e Huber (2025) confirmam esse comportamento em 18 modelos distintos e introduzem o conceito de *context rot*, em que a degradação se aprofunda progressivamente à medida que distrações se acumulam no contexto, antes mesmo de o modelo aproximar seu limite de tokens. Os autores identificam que a degradação começa a partir de aproximadamente 40% de uso da janela de contexto e se torna colapso a partir de 60%, onde o modelo passa a ignorar instruções, repetir erros e alucinar com confiança, conforme ilustrado na Figura 12. Esse resultado descarta a ideia de que janelas de contexto maiores resolvem o problema de memória.

Figura 12 – Zonas de desempenho em função do uso da janela de contexto.



Fonte: Elaborado pelo autor, adaptado de Hong, Troynikov e Huber (2025).

Adicionar memória ao agente, porém, não elimina o problema por si só. Uma implementação mal projetada introduz falhas próprias. Se a recuperação trazer fatos irrelevantes para a consulta atual, o agente passa a responder com base no que está armazenado em vez do que foi perguntado (DU, P., 2026). Resumos repetidos ao longo de sessões longas podem distorcer gradualmente os fatos originais, e o agente começa a trabalhar com um histórico corrompido sem perceber. Fatos desatualizados, armazenados sem controle de validade, continuam sendo recuperados e tratados como verdade atual. Nesses casos, ter memória é pior do que não ter porque fazem com que o agente erre com mais confiança.

O problema, portanto, envolve diversos aspectos, como a seleção, a compactação e a recuperação. Sem um mecanismo que decida o que preservar, o que descartar e o que recuperar em cada turno, o agente acumula ruído, perde continuidade e tende a respostas inconsistentes. Para a LivIA, isso se traduz em requisitos concretos. É

preciso manter coerência ao longo de múltiplos turnos dentro de uma sessão, persistir informações relevantes entre sessões distintas e recuperar fatos operacionais com precisão, sem aumentar o custo de tokens por consulta.

3.3 REQUISITOS TÉCNICOS

Os requisitos a seguir foram definidos a partir do problema descrito na Seção 3.2. Eles organizam o que o sistema deve fazer e as condições que a solução precisa atender para ser considerada adequada ao contexto de uso.

3.3.1 Requisitos Funcionais

Os requisitos funcionais cobrem os dois módulos da solução: a memória de sessão e a memória de longo prazo.

Quadro 1 – Requisitos funcionais do sistema.

Código	Descrição
RF-01	Buffer de histórico — o sistema deve manter o histórico da interação corrente em um buffer e disponibilizá-lo ao agente a cada turno.
RF-02	Compactação por limite de tokens — quando o buffer ultrapassar um limite configurável de tokens, o sistema deve compactar o histórico e criar um resumo da interação atual.
RF-03	Segmentação por tópico — a compactação deve ocorrer em fronteiras de mudança de tópico, e não por truncagem fixa.
RF-04	Sumário Estruturado de Sessão — o sistema deve manter um sumário estruturado de sessão consolidando objetivo da sessão e informações importantes, evitando <i>goal drift</i> em sessões longas.
RF-05	Extração de fatos durante a compactação — durante a compactação do histórico, o sistema deve extrair fatos do usuário (preferências, papel, setor) e fatos do ambiente (entidades operacionais citadas) para persistência de longo prazo.
RF-06	Persistência relacional — fatos extraídos devem ser persistidos em banco relacional, indexados por <i>user_id</i> , <i>company_id</i> e entidade.
RF-07	Recuperação em nova sessão — em nova sessão do mesmo usuário, o agente deve recuperar fatos relevantes e injetá-los no contexto antes de gerar a primeira resposta.
RF-08	Isolamento por usuário e empresa — o sistema não deve recuperar fatos de um usuário ou empresa ao atender outro.
RF-09	Atualização por contradição — quando uma nova informação contradiz fato já registrado, o sistema deve atualizar o fato e a versão mais recente prevalece.
RF-10	Deduplicação na admissão — o sistema deve deduplicar memórias semanticamente equivalentes no momento da admissão (controle de admissão).

Fonte: Elaborado pelo autor.

3.3.2 Requisitos Não-Funcionais

Os requisitos não-funcionais definem as condições de qualidade, desempenho e operação que a solução precisa respeitar.

Quadro 2 – Requisitos não-funcionais do sistema.

Código	Descrição
RNF-01	Eficiência de contexto — os tokens de entrada por turno não devem crescer linearmente com o número de turnos em sessões longas.
RNF-02	Latência — a latência p95 da resposta não deve degradar significativamente em relação ao baseline sem memória.
RNF-03	Custo — o custo médio por turno deve reduzir em relação ao baseline com histórico completo.
RNF-04	Veracidade — a taxa de alucinação não deve aumentar com memória ativa em relação ao baseline.
RNF-05	Privacidade e isolamento — as memórias devem ser estritamente segregadas por <code>user_id</code> e <code>company_id</code> .
RNF-06	Idempotência da extração — extrações repetidas sobre o mesmo conteúdo não devem produzir registros duplicados na memória de longo prazo.
RNF-07	Processamento assíncrono — a persistência da memória de longo prazo devem ocorrer de forma assíncrona, sem bloquear a resposta ao usuário.

Fonte: Elaborado pelo autor.

Com o problema delimitado e os requisitos definidos, o Capítulo 4 apresenta a solução proposta, descrevendo a arquitetura do sistema e as decisões técnicas que orientam cada módulo de memória.

4 SOLUÇÃO PROPOSTA E METODOLOGIA UTILIZADA

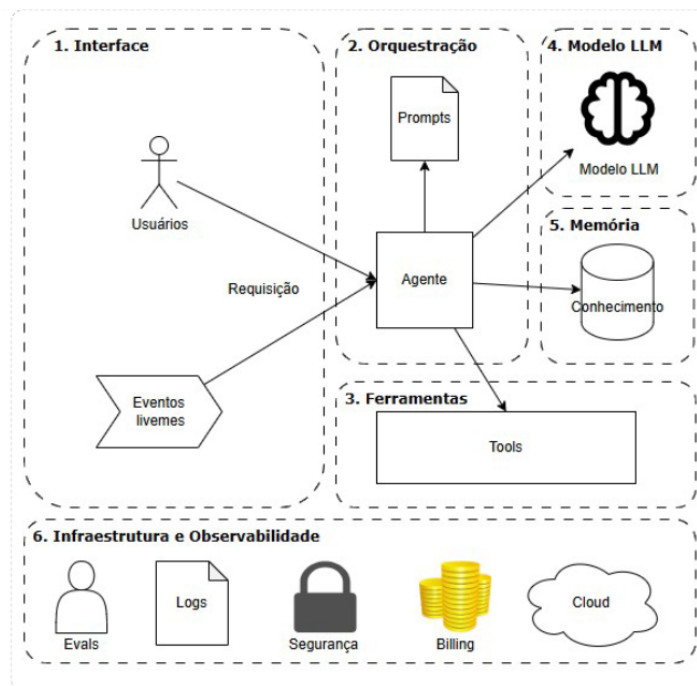
Este capítulo apresenta a solução proposta para o problema descrito no capítulo anterior, detalhando a arquitetura do sistema e os módulos de memória que a compõem.

A Seção 4.1 apresenta uma visão geral da arquitetura, mostrando como os componentes se conectam e como uma conversa flui dentro do sistema. A Seção 4.2 detalha o módulo de memória de sessão, responsável pelo gerenciamento do contexto durante a interação atual. Por fim, a Seção 4.3 descreve o módulo de memória de longo prazo, que persiste as informações entre as sessões.

4.1 VISÃO GERAL DA ARQUITETURA DO SISTEMA

LIVIA é um agente de IA voltado ao atendimento de operadores industriais de múltiplas empresas. Sua arquitetura organiza-se em camadas, conforme a Figura 13: a Interface recebe as mensagens e os eventos da plataforma, a Orquestração monta o *prompt* e coordena o fluxo, o Modelo LLM gera as respostas, as Ferramentas (*tools*) dão acesso aos dados operacionais, a Memória armazena e recupera o conhecimento acumulado e a Infraestrutura e Observabilidade sustenta a operação.

Figura 13 – Arquitetura geral em camadas do agente.



Fonte: Elaborado pelo autor.

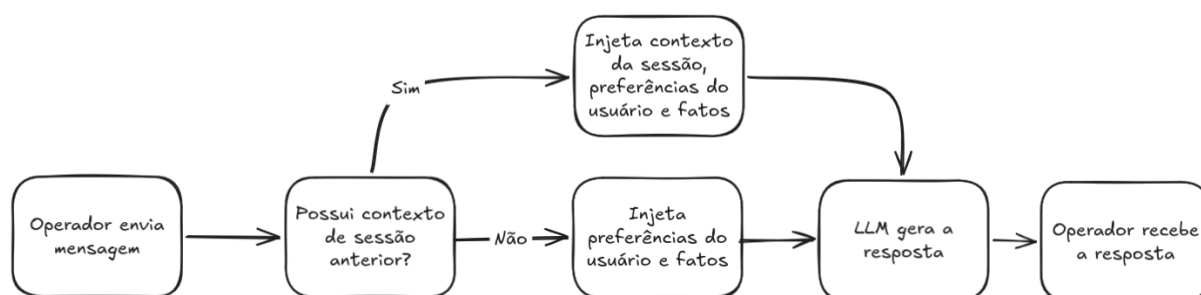
Este trabalho concentra-se na camada de Memória, responsável por preservar o conhecimento entre turnos e entre sessões. Sobre ela, são propostos dois módulos

complementares. A memória de sessão, de curto prazo, mantém o contexto da interação corrente dentro da janela de contexto do modelo, combinando *buffer* com janela deslizante, sumarização em um sumário estruturado e segmentação dinâmica por tópico (Seção 4.2). A memória factual, de longo prazo, persiste preferências do usuário e fatos do ambiente entre sessões, por meio de destilação estruturada, controle de admissão semântica e recuperação híbrida (Seção 4.3).

A operação desses módulos organiza-se em torno de uma separação fundamental: o caminho síncrono, que entra na latência percebida pelo operador, e o caminho assíncrono, que executa as operações de memória após a resposta ser enviada, sem bloquear o próximo turno.

Ao receber a mensagem do operador, o sistema verifica se há contexto ativo para aquela sessão. Se houver, recupera o sumário da sessão corrente junto com as preferências do usuário e os fatos do ambiente. Se não houver, recupera apenas preferências e fatos, se existirem. Com o *prompt* montado, o agente consulta o modelo de linguagem e envia a resposta ao operador. A Figura 14 ilustra esse fluxo.

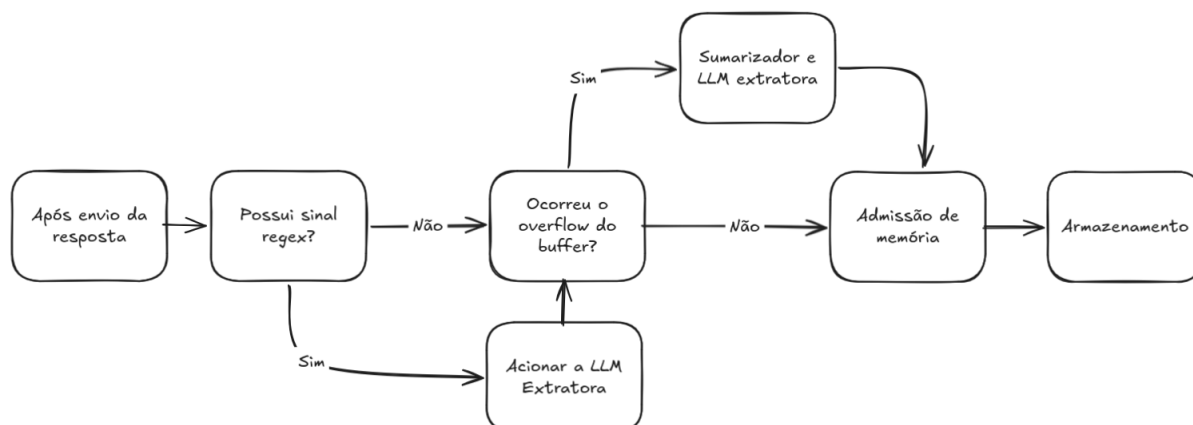
Figura 14 – Fluxo de um turno: decisão de contexto e envio da resposta.



Fonte: Elaborado pelo autor.

Após o envio, o caminho assíncrono é ativado. A mensagem do operador é avaliada por expressões regulares (*regex*), sequências de caracteres que definem um padrão de busca, capazes de detectar marcadores de memórias duráveis como “prefiro”, “sempre” ou “nunca”. Se algum padrão dispara, o extrator é acionado em segundo plano. Em seguida, o sistema verifica se o buffer atingiu o limite. Se sim, o Compactador sumariza o histórico e aciona o extrator sobre a janela completa. As duas etapas são independentes: uma falha não cancela a outra, e ambas disparam de forma assíncrona. A Figura 15 detalha esse caminho.

Figura 15 – Caminho assíncrono de memória após o envio da resposta.



Fonte: Elaborado pelo autor.

A Seção 4.2 detalha o módulo de memória de sessão, com o pipeline de compactação, o formato do sumário estruturado e a segmentação por tópico. A Seção 4.3 descreve o módulo de memória de longo prazo, com a destilação estruturada, a admissão semântica e a recuperação híbrida.

4.2 MÓDULO DE MEMÓRIA DE SESSÃO

O módulo de memória de sessão é responsável por manter o contexto da interação corrente dentro dos limites da janela de contexto do modelo. Para isso, atua em três frentes. A Seção 4.2.1 descreve o pipeline de compactação, que controla o crescimento do buffer e aciona a sumarização assíncrona quando for necessário. A Seção 4.2.2 apresenta o sumário estruturado de sessão, o formato XML que o sumarizador produz e que representa o contexto resumido da sessão atual. Por fim, a Seção 4.2.3 explica como o sistema detecta mudanças de tópico e decide quando criar um novo registro de sessão no banco de dados.

4.2.1 Pipeline de compactação do histórico

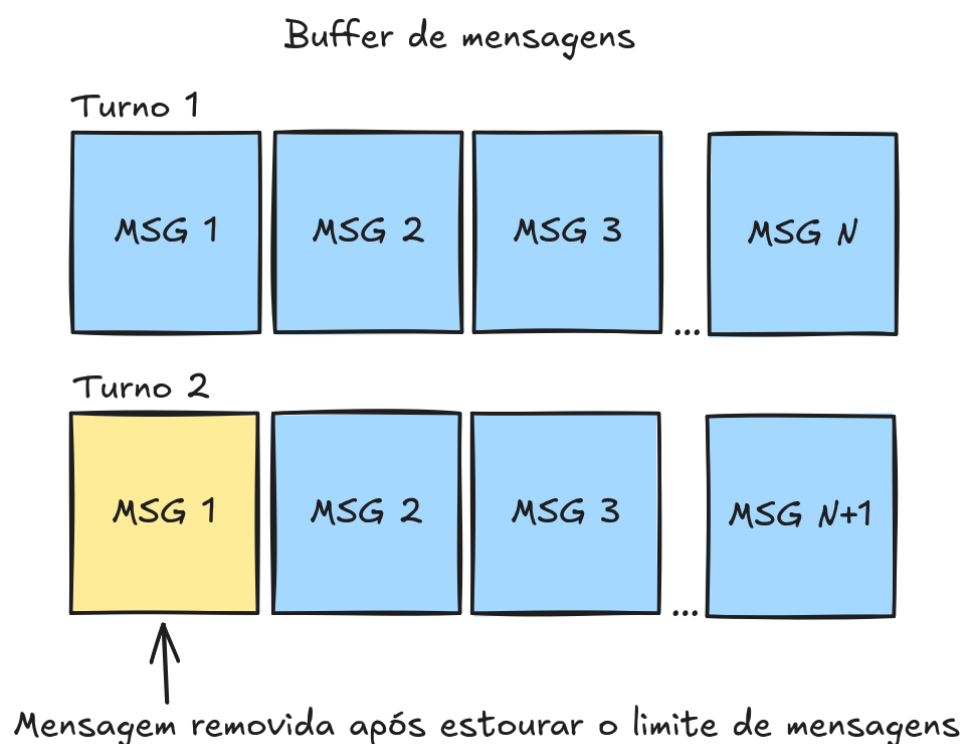
O Compactador é o módulo responsável pelo gerenciamento do contexto durante a sessão. Ele atua em duas etapas sequenciais, o *buffer* de mensagens e o resumo do histórico.

O buffer retém as N mensagens mais recentes da interação, onde N é configurável. Para manter esse limite, o sistema aplica a técnica de janela deslizante (*sliding window*), onde conforme novas mensagens chegam, as mais antigas são removidas. Dessa forma, o agente sempre opera sobre um histórico recente e coerente, sem que o contexto cresça sem controle.

A Figura 16 detalha a janela deslizante atuando sobre o buffer ao longo de dois turnos. Os blocos em azul são as mensagens que compõem a janela ativa, mantidas no

contexto do agente, e o bloco em amarelo é a mensagem mais antiga, que sai da janela quando uma nova mensagem chega no Turno 2. Esse deslocamento mantém no buffer apenas as N mensagens mais recentes. A mensagem que sai não é simplesmente descartada, ela é utilizada pelo sumariizador, que consolida seu conteúdo no sumário estruturado de sessão e persiste o resultado no banco de dados.

Figura 16 – Buffer de mensagens com janela deslizante.



Fonte: Elaborado pelo autor.

A compactação é ativada por dois gatilhos, o limite de N mensagens ou o limite de X tokens, o que for atingido primeiro. Quando isso ocorre, o sistema gera um resumo de todo o conteúdo acumulado até aquele momento. Esse resumo é produzido por uma chamada dedicada a um modelo de linguagem com *prompt* especializado, separado do modelo principal do agente. O conteúdo e a estrutura desse *prompt* são detalhados no Capítulo 5.

Para não aumentar o tempo de resposta ao operador, a sumarização ocorre de forma assíncrona. A cada interação, o operador envia uma mensagem, ela é adicionada ao buffer e o agente gera a resposta normalmente. Ao final, o sistema verifica o tamanho do buffer e a contagem de tokens. Se nenhum limite for atingido, nada acontece. Se algum for, a sumarização é disparada em segundo plano, sem bloquear o operador. O resumo gerado segue o formato de sumário estruturado de sessão, detalhado na Seção 4.2.2. Em seguida, ele substitui o anterior no armazenamento temporário e uma cópia é persistida no banco de dados.

4.2.2 Sumário Estruturado de Sessão

Em vez de acumular o histórico bruto entre turnos, o sistema consolida o estado da sessão em um bloco XML estruturado, que será chamado aqui de sumário estruturado de sessão. A abordagem é inspirada no MEM1 (ZHOU *et al.*, 2025), que propõe que o agente mantenha um estado compacto aglutinando memória com o raciocínio de cada turno descartando o histórico anterior, em favor de uma representação compacta do que importa. No MEM1, esse comportamento é obtido por treinamento do modelo. Neste trabalho, não foi realizado nenhum treinamento. Apenas o conceito de estado compacto foi adotado e adaptado para um formato que atenda ao contexto da sessão do operador.

O bloco é composto por cinco campos. Dois descrevem o foco da sessão, o *title*, que registra em três a cinco palavras o tópico central, e o *intent*, que contém o objetivo principal em que a conversa está seguindo. O *key_facts* consolida os fatos e as conclusões encontrados ao longo da interação, resumindo grandes volumes de informações em descrições breves e compactas. O *decisions* as registra escolhas feitas e as conclusões alcançadas. O quinto campo, *continuity*, não armazena conteúdo da conversa, mas sinaliza se o tópico continuou ou não, sendo usado pela segmentação descrita na Seção 4.2.3.

O campo *intent* não atua como uma restrição ao comportamento do agente, mas sim como uma referência de contexto. Quando uma pergunta admite mais de uma interpretação, o agente tende a escolher a que melhor se alinha ao objetivo registrado, ajudando a manter a consistência ao longo da sessão sem que engesse a conversa.

O Quadro 3 apresenta a estrutura do bloco XML gerado pelo sumarizador.

Quadro 3 – Estrutura do sumário estruturado de sessão em XML.

```
<summary>
  <title>Tópico central da sessão (3 a 5 palavras)</title>
  <continuity>continuous | discontinuous</continuity>
  <intent>Objetivo principal do operador na sessão</intent>
  <key_facts>Fatos e conclusões relevantes acumulados</key_facts>
  <decisions>Escolhas feitas e conclusões alcançadas</decisions>
</summary>
```

Fonte: Elaborado pelo autor.

A escolha de XML em vez de JSON foi proposital. Modelos de linguagem produzem XML com maior fidelidade estrutural porque as tags nomeadas reduzem erros de formatação, mais comuns em JSON quando o conteúdo contém aspas ou quebras de linha. Os campos *key_facts* e *decisions* costumam ter múltiplas frases, o que tornaria o escape em JSON propenso a falhas. Além disso, a extração dos campos é simples e não depende de um parser que quebre em qualquer caractere mal formado. A cada iní-

cio de sessão, o sumário estruturado é injetado no *system prompt* do agente como um bloco [SESSION CONTEXT]. Dessa forma, o agente retoma a conversa com o contexto compactado, sem precisar recarregar o histórico completo. O sumário é atualizado a cada compactação e persiste no banco de dados entre sessões.

O sumário é regenerado a cada compactação. O sumarizador recebe as N mensagens do buffer junto com o resumo anterior e produz um novo bloco XML que reflete o estado atual da conversa. Em vez de acumular os campos de forma isolada, o sistema reconstrói o resumo com base em tudo que foi dito até aquele momento, atualizando o que mudou e preservando o que ainda é relevante. O campo *continuity* orienta o que acontece com o registro gerado, conforme detalhado na Seção 4.2.3.

4.2.3 Controle de Continuidade de Tópico

Uma abordagem comum para controlar o crescimento do histórico é descartar as mensagens mais antigas quando o buffer atinge seu limite, sem considerar o que está sendo discutido. Essa truncagem fixa ignora completamente as fronteiras de tópico. Uma sessão pode trocar de assunto várias vezes ao longo dos turnos, e cortar por uma quantidade fixa pode fragmentar um contexto importante no meio, assim errando resumos que misturam tópicos distintos e comprometem a coerência do agente.

Para resolver isso, foi adotada uma segmentação dinâmica baseada em tópicos, inspirada no Topic Loom do MemBox (TAO *et al.*, 2026). No MemBox, o Topic Loom monitora o diálogo a cada nova mensagem, consultando um LLM para classificar se a conversa continua no mesmo tópico ou se mudou de assunto. Neste trabalho, essa classificação foi adaptada para ocorrer durante a compactação. Com isso, o próprio sumarizador, ao gerar o bloco XML estruturado, inclui o campo *continuity* com a avaliação de continuidade em relação ao resumo anterior. Dessa forma, a decisão de segmentação não adiciona uma chamada extra ao LLM, aproveitando a sumarização que já ocorre.

Para suportar múltiplos tópicos por sessão sem sobrescrever todo o histórico, foi criado o mecanismo de *topic_sequence*. Cada sessão pode ter mais de uma linha na tabela de registros, identificadas por um contador que incrementa a cada novo tópico. Quando a classificação é *continuous*, o sistema atualiza a linha atual com o novo resumo. Quando é *discontinuous*, uma nova linha é inserida com o *topic_sequence* incrementado e a linha anterior é mantida intacta, preservando o contexto passado. Dessa forma, o agente mantém um registro separado de cada tópico tratado na sessão, sem que um novo assunto sobrescreva o histórico de um anterior.

Para ilustrar o mecanismo, considere uma sessão em que o operador discute o OEE de uma máquina e depois muda de assunto para uma dúvida sobre cadastro de usuário. Após a primeira compactação, o sumarizador classifica a conversa como *continuous* e o sistema insere a primeira linha com *topic_sequence* igual a 1, conforme

o Quadro 4.

Quadro 4 – Estado da tabela após a primeira compactação (*continuous*).

topic_sequence	title
1	Análise OEE CT01

Fonte: Elaborado pelo autor.

Na segunda compactação, o sumariizador avalia que a conversa ainda se trata do mesmo assunto e a classifica como *continuous*. O sistema faz *upsert*, uma operação que insere um registro caso ele não exista e atualiza caso já esteja presente no banco, na linha existente, atualizando o resumo sem criar uma nova entrada, como mostra o Quadro 5.

Quadro 5 – Estado da tabela após a segunda compactação (*continuous* com *upsert*).

topic_sequence	title
1	Causas de Queda de OEE CT01

Fonte: Elaborado pelo autor.

Na terceira compactação, o operador muda de assunto e o sumariizador classifica como *discontinuous*. O sistema insere uma nova linha com *topic_sequence* igual a 2 e preserva a anterior intacta, conforme o Quadro 6.

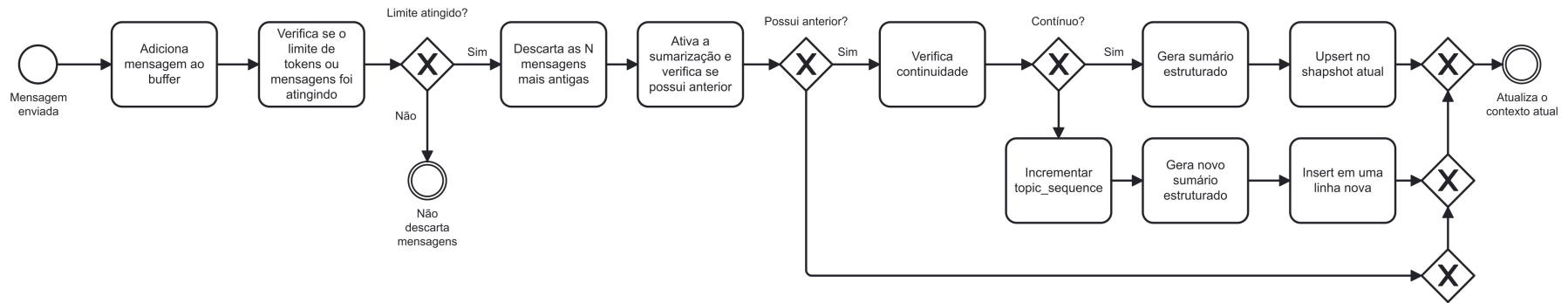
Quadro 6 – Estado da tabela após a terceira compactação (*discontinuous* com *insert*).

topic_sequence	title
1	Causas de Queda de OEE CT01
2	Cadastro de Usuário no Sistema

Fonte: Elaborado pelo autor.

A Figura 17 ilustra o fluxo completo do pipeline de compactação, desde a detecção do overflow até a persistência do registro com a decisão de continuidade.

Figura 17 – Fluxo do pipeline de compactação e segmentação por tópico.



Fonte: Elaborado pelo autor.

Uma regra adicional previne a fragmentação antecipada. Quando o buffer ainda não atingiu seu limite, o sistema ignora o resultado da classificação e força a atualização da linha atual. Isso evita que trocas de assunto em turnos iniciais criem novos tópicos com conteúdo insuficiente para gerar um resumo informativo, o que só causaria o acúmulo de informações irrelevantes ao banco de dados. A implementação detalhada desse mecanismo é apresentada no Capítulo 5.

4.3 MÓDULO DE MEMÓRIA DE LONGO PRAZO

Sem memória de longo prazo, todo o conhecimento construído ao longo de uma sessão se perde ao fim de cada interação. O operador que ensinou ao agente seu setor de atuação, seu turno ou suas preferências de formato precisaria repetir essas instruções na próxima interação, como se a conversa anterior nunca tivesse acontecido. O módulo de memória de longo prazo resolve esse problema persistindo conteúdos relevantes entre sessões e os disponibilizando ao agente no início de cada nova interação. As informações persistidas se dividem em dois tipos de Memória Factual, sendo elas: as preferências do usuário, que registram como o operador quer interagir com o agente, e os fatos do ambiente, que descrevem o contexto em que está inserido. A Seção 4.3.1 aprofunda cada categoria de memória e como são estruturadas. A Seção 4.3.2 descreve como essas memórias são formadas a partir das interações com o agente. A Seção 4.3.3 trata da recuperação, e a Seção 4.3.4 detalha o mecanismo que controla a admissão de memórias na base.

4.3.1 Preferências do Usuário e Fatos do Ambiente

Em vez de armazenar apenas informações brutas coletadas, o sistema converte mensagens relevantes em registros estruturados, seguindo o princípio proposto por Lewis (2026): representar cada informação extraída em um objeto que captura o essencial, descartando o que é redundante ou transitório.

A parte do usuário captura as preferências de como o operador quer interagir com o sistema, não apenas o que foi pedido em um turno específico, mas padrões estáveis de estilo ou comunicação que devem ser mantidos ao longo das sessões. Alguns exemplos como: qual o idioma preferido para as respostas, o formato que facilita a leitura de indicadores ou o nível de detalhamento esperado em cada mensagem. Cada preferência é representada por quatro campos, sendo eles: O *exchange_core*, que registra a essência da preferência como uma instrução imperativa de até 80 caracteres. O *specific_context*, que delimita quando ela se aplica, sendo nulo quando não tiver uma restrição de contexto. O *room*, que classifica o domínio da preferência, tendo seu nome inspirado na arquitetura de palácio de memória (*memory palace*) proposta por Lewis (2026), em que cada registro é atribuído a um quarto temático que

agrupa informações relacionadas. Para decidir esses quartos, o extrator, detalhado na Seção 4.3.2, é orientado a usar *communication*, *language* ou *format*, vocabulário que cobre naturalmente o espaço de classificação das interações. O *source*, usado apenas para diferenciação no banco de dados, registra se a extração ocorreu durante um turno ou durante a compactação.

Os fatos do ambiente descrevem o contexto operacional do operador, como o setor que supervisiona, o turno em que trabalha ou os equipamentos que acompanha com maior frequência. A estrutura é idêntica à das preferências, com os mesmos quatro campos, porém o *room* tem outras classificações, com as orientações para o extrator separar da seguinte forma: *sector*, *shift*, *role* e *equipment*. Ambos compartilham o mesmo campo livre. A distinção é de orientação ao extrator, não de esquema. Apesar da forma idêntica, preferências e fatos são armazenados em tabelas distintas e injetados em blocos separados no *prompt*, [USER PREFERENCES] e [ENVIRONMENT FACTS]. Por terem a mesma estrutura, os dois tipos reusam o mesmo pipeline de extração e a mesma política de admissão, sem que seus conteúdos se misturem.

Os campos *exchange_core*, *specific_context* e *room* foram adotados diretamente de Lewis (2026). O formato de injeção no *prompt*, mostrado no Quadro 7, é uma adaptação deste trabalho para disponibilizar as memórias ao agente a cada sessão. O Quadro 8 apresenta exemplos de cada tipo.

Quadro 7 – Formato de representação das memórias de longo prazo.

Variante	Formato
Sem restrição de contexto	(room) exchange_core
Com restrição de contexto	(room) exchange_core when: specific_context

Fonte: Elaborado pelo autor.

Quadro 8 – Exemplos de representação de preferências do usuário e fatos do ambiente.

Tipo	room	Representação
Preferência	language	(language) Responda sempre em português
Preferência	format	(format) Use duas casas decimais para OEE when: análise comparativa
Fato do ambiente	sector	(sector) Trabalha no setor Compressoras
Fato do ambiente	shift	(shift) Trabalha no turno 2 when: dias de semana

Fonte: Elaborado pelo autor.

4.3.2 Destilação Estruturada

Ainda inspirado em Lewis (2026), este trabalho adota a destilação estruturada como processo de produção desses registros. A extração é ativada por dois gatilhos. O primeiro ocorre após cada resposta do agente, onde a mensagem do operador é avaliada por dois sinais léxicos baseados em expressões regulares (*regex*), independentes entre si, um voltado às preferências do usuário e o outro aos fatos do ambiente. Se ao menos um disparar, uma tarefa assíncrona aciona o LLM extrator com uma única chamada, que classifica e separa os dois tipos pelo campo *kind*. As memórias geradas por esse caminho são registradas com *source* igual a *heuristic_turn*, refletindo que a ativação se deu pelo sinal *regex* no turno. O segundo gatilho ocorre durante a compactação do histórico. Quando o pipeline descrito na Seção 4.2.1 gera um novo resumo, a extração roda sobre a janela completa de mensagens. Esse caminho cobre preferências e fatos que o *regex* pode não ter detectado por turno. As memórias geradas aqui recebem *source* igual a *compaction*.

No caminho de compactação, a extração é vinculada à geração de um novo resumo, então se o sumário não produz um bloco novo, a extração não é acionada. Quando a compactação ocorre, são feitas duas chamadas em sequência a modelos distintos, o sumário e o extrator, em chamadas separadas, nunca unidas em um único *prompt*. O extrator retorna preferências e fatos em uma única passada, de modo que o caminho de compactação usa no máximo dois modelos por ciclo.

Os padrões de classificação de cada categoria têm vocabulários distintos: os de preferência carregam verbos de postura como "prefiro", "sempre" e "nunca", enquanto os de fato do ambiente carregam termos do contexto operacional como setor, turno e referências a ordens de produção. A separação permite que cada subsistema dispare de forma independente. Quando nenhum sinal está presente, o turno não aciona nenhuma chamada ao LLM.

Para reduzir a geração de candidatos redundantes antes da admissão, o extrator recebe as memórias já registradas para aquele operador e é instruído a extrair apenas itens genuinamente novos. Esse mecanismo constitui uma primeira linha de deduplicação no nível da linguagem, complementar à deduplicação semântica descrita na Seção 4.3.4.

O extrator é também instruído a não confundir memória durável com perguntas pontuais sobre dados da operação. Expressões como "qual o OEE do CT01 hoje?" descrevem uma requisição pontual, não um fato do operador. Sem essa instrução, qualquer consulta sobre dados recorrentes poderia ser interpretada como fato e contaminar a base com informações que não se sustentam entre sessões, comportamento observado durante o desenvolvimento e que motivou refinamentos no *prompt* do extrator.

O termo "orientado", usado ao descrever as classificações de *room* na Se-

ção 4.3.1, é intencional. Por se tratar de um modelo de linguagem, o extrator não oferece garantia absoluta de que irá classificar os itens exatamente conforme instruído. O desvio é raro na prática, mas não impossível.

4.3.3 Persistência e Recuperação Híbrida

O sistema mantém dois tipos de memória persistente: os registros de sessão gerados pela sumarização e as preferências e fatos do ambiente extraídos pelo módulo de longo prazo. Essas memórias são recuperadas de três formas complementares.

O primeiro caminho passivo recupera o resumo da sessão atual descrito na Seção 4.2.2. O resumo é mantido num cache de curto prazo durante a sessão, é desse cache que o bloco [SESSION CONTEXT] é montado a cada turno. Quando esse cache expira ou não está disponível, o sistema busca no banco de dados o registro mais recente do operador. Se o registro é do mesmo dia, o contexto é recuperado e injetado no *prompt* como [SESSION CONTEXT], retomando a sessão de onde parou. Se é de um dia anterior, a sessão começa vazia. Como o identificador de sessão é composto pelo par operador/empresa, qualquer nova janela aberta pelo mesmo operador no mesmo dia retoma o mesmo contexto.

O segundo caminho passivo entrega as preferências do usuário e os fatos do ambiente no início de cada turno, com os blocos [USER PREFERENCES] e [ENVIRONMENT FACTS] inseridos no *prompt* do sistema.

O terceiro caminho é ativo. Quando o operador referencia uma conversa anterior com expressões como “o que discutimos ontem” ou “lembra da análise do CT01 ontem”, o agente chama a ferramenta *search_session_history*. Essa ferramenta realiza uma busca textual sobre os títulos dos resumos registrados, filtrando por similaridade léxica com o termo buscado, por data ou pela combinação dos dois. A busca é léxica porque o operador tende a citar termos específicos como um código de centro de trabalho ou uma data, para os quais a correspondência de texto é mais eficaz do que a proximidade de *embedding*. Quando nenhum critério é fornecido, a ferramenta retorna os registros mais recentes. Referências a conversas passadas quase sempre carregam uma referência temporal, como “ontem”, “semana passada” ou “quando discutimos o CT01”, tornando a data um critério natural de filtragem. Essa característica separa a ferramenta das consultas operacionais: “qual análise fizemos sobre o CT01 ontem” é uma busca no histórico de sessão, agora “qual o OEE do CT01 esse mês” é uma consulta a dados da plataforma. O agente tem acesso a várias ferramentas, e a descrição de cada uma delimita quando deve ser usada, guiando a seleção e reduzindo erros de uso incorreto.

A divisão entre os três caminhos reflete algumas naturezas distintas da informação. O resumo da sessão é volátil e delimitado ao dia. Preferências e fatos do ambiente são estáveis, sempre relevantes e recuperados de forma integral. Enquanto, explica-

ções a partir de resumos de sessões passadas possuem relevância dependendo do contexto e são buscados apenas sob demanda.

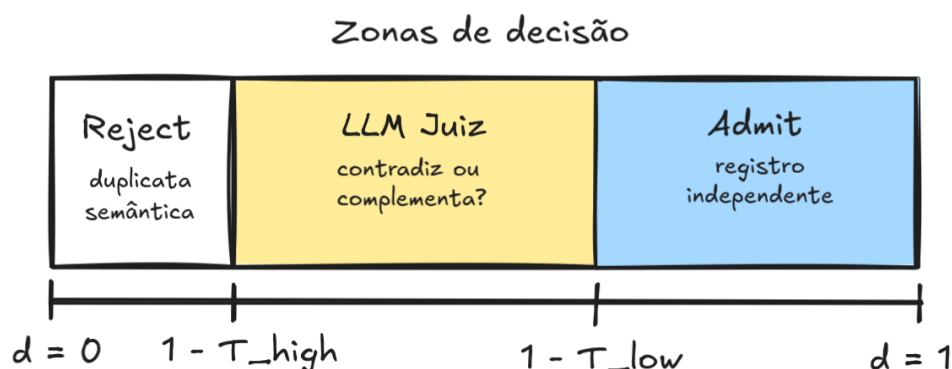
Os dois caminhos passivos injetam conteúdo diretamente no *prompt* a cada turno. Isso contrasta com o problema identificado no Capítulo 3, que contextos extensos aumentam custo, latência e degradam a qualidade das respostas. A injeção direta é viável aqui porque o conteúdo é curado e compacto, não se trata do histórico bruto, mas da informação destilada. O limite configurável de preferências e fatos admitidos garante que esses blocos não cresçam sem controle, tornando a injeção direta o *trade-off* adequado entre disponibilizar o contexto do operador e não inflar o *prompt*.

4.3.4 Controle de Admissão de Memória

Sem controle sobre o que entra na base, o módulo de memória tende a acumular registros redundantes ou inconsistentes, inflando os blocos injetados no *prompt* e degradando RNF-01 e RNF-03. Para evitar esse comportamento, cada candidato a memória passa por uma política de admissão antes de ser persistido, inspirada no trabalho de Guilin Zhang *et al.* (2026), que resulta em uma de três decisões: *REJECT*, *UPDATE* ou *ADMIT*.

A admissão opera em duas camadas. A primeira camada gera o vetor semântico do candidato e mede a distância cosseno em relação ao vizinho mais próximo já armazenado para aquele operador, onde distância pequena indica alta semelhança e distância grande indica independência semântica. Essa distância é comparada a dois limiares, T_{high} e T_{low} , definidos em similaridade cosseno. Se a distância for no máximo $1 - T_{high}$, o candidato é tratado como equivalente a uma memória existente e recebe *REJECT*. Se for maior que $1 - T_{low}$, o candidato é considerado independente e recebe *ADMIT* diretamente. Entre os dois limiares, a distância sozinha não é suficiente para decidir e a segunda camada é acionada. Dessa forma, apenas a região incerta paga o custo de uma chamada ao LLM, enquanto os dois extremos são resolvidos apenas com a distância. A Figura 18 ilustra as três zonas sobre o eixo de distância.

Figura 18 – Zonas de decisão da política de admissão de memória.



Fonte: Elaborado pelo autor.

A segunda camada é um classificador LLM com temperatura zero, parâmetro que reduz a aleatoriedade da geração e deixa a saída mais determinística, embora não garanta que a mesma entrada produza sempre a mesma saída em chamadas futuras. O classificador avalia a relação semântica entre o candidato e seu vizinho mais próximo. Se o candidato contradiz a memória existente, a entrada antiga é substituída pela nova, seguindo a semântica de *last-write-wins*, em que a versão mais recente substitui a anterior (RF-09). Se a relação for de complementaridade ou independência, ambas coexistem. Quando nenhum classificador está configurado, a decisão padrão na faixa intermediária é *ADMIT*, preservando a memória sem exigir o LLM. Se a chamada ao classificador falhar, o candidato é descartado e o turno segue normalmente. Perder um candidato válido é preferível a admitir informação possivelmente redundante ou contraditória, o que degradaria a qualidade do contexto injetado.

A distância cosseno foi adotada como critério da primeira camada porque captura similaridade semântica independente da forma do texto, conforme definido na Seção 2.5.3.3. Essa propriedade é exatamente o que a admissão requer: “seja conciso nas respostas” e “responda de forma breve e direta” devem ser detectadas como equivalentes, mesmo sem compartilhar nenhuma palavra. A admissão é agnóstica ao tipo de memória: a mesma política serve preferências e fatos do ambiente por meio do protocolo compartilhado descrito na Seção 4.3.1. Os limiares T_{high} e T_{low} são definidos a partir de pares representativos, com método e valores apresentados na Seção 5.4.1.

A solução descrita neste capítulo foi implementada de forma concreta no Capítulo 5, que detalha as escolhas de infraestrutura, as bibliotecas utilizadas e os resultados obtidos a partir dos critérios de avaliação definidos.

5 DESENVOLVIMENTO E ANÁLISE DOS RESULTADOS OBTIDOS

Este capítulo descreve a implementação concreta da solução apresentada no capítulo anterior, as decisões arquiteturais que levaram à arquitetura final e a análise dos resultados obtidos. As escolhas de bibliotecas, ferramentas e infraestrutura são justificadas em função dos requisitos funcionais e não funcionais definidos no Capítulo 3.

A Seção 5.1 apresenta o ambiente de execução, as tecnologias utilizadas e como os componentes se conectam. Em seguida, a Seção 5.2 e a Seção 5.3 detalham a implementação de cada módulo de memória, incluindo as decisões tomadas ao longo do desenvolvimento. Por fim, a Seção 5.4 avalia em que medida os requisitos foram atendidos e discute os resultados de eficiência obtidos.

5.1 IMPLEMENTAÇÃO DO AMBIENTE E INFRAESTRUTURA

A implementação do sistema LivIA é composta por cinco componentes tecnológicos principais, escolhidos com base nos requisitos funcionais e não funcionais definidos no Capítulo 3. O Quadro 9 apresenta cada componente, a tecnologia adotada e o papel que desempenha no sistema.

Quadro 9 – Componentes tecnológicos do sistema LivIA.

Componente	Tecnologia	Papel no sistema
Orquestração do agente	<i>Framework</i> de orquestração de agentes	Ciclo ReAct, gerenciamento do buffer de mensagens
Contexto ativo	Cache em memória	Cache de sessão com expiração por inatividade
Armazenamento durável	Banco de dados relacional com extensão vetorial	Registros de sessão, preferências e fatos do ambiente; busca vetorial (índice HNSW, distância de cosseno) para a admissão semântica
Modelo de linguagem	Modelo de linguagem adotado	Sumarização, extração e geração de respostas
Embeddings semânticos	Modelo de <i>embeddings</i> adotado	Vetorização para admissão semântica

Fonte: Elaborado pelo autor.

A comunicação com o cliente usa o protocolo AG-UI com *Server-Sent Events* (SSE), que transmite tokens do agente ao operador à medida que são gerados (AG-UI-

PROTOCOL, 2025). O AG-UI é um protocolo aberto voltado à integração entre agentes e interfaces de usuário, com suporte a personalizações de *frontend* como *widgets*. Por atuar como assistente virtual do operador com interação contínua, o LIVIA se enquadra diretamente no perfil de uso para o qual o AG-UI foi projetado.

O *framework* de orquestração de agentes adotado coordena o ciclo entre o LLM e as ferramentas e oferece integração nativa com Protocolo de Contexto de Modelo (*Model Context Protocol*) (MCP), protocolo padronizado para comunicação entre agentes e ferramentas externas, eliminando a necessidade de adaptadores customizados. O gerenciamento do buffer de mensagens inicialmente implementado de forma manual foi posteriormente migrado para a implementação nativa do *framework*. A sumarização, sem equivalente nativo, permanece implementada diretamente no sistema, o que mantém controle total sobre o formato e os campos do sumário estruturado de sessão.

O Quadro 10 lista os parâmetros configuráveis do sistema, referenciados nas seções seguintes conforme cada mecanismo é descrito.

Quadro 10 – Parâmetros configuráveis do sistema de memória.

Parâmetro	Descrição
<code>window_size</code>	Tamanho máximo do buffer de mensagens
<code>threshold_tokens</code>	Limite para acionar a compactação, com margem de 15% aplicada
<code>keep_last_n</code>	Mensagens recentes preservadas verbatim na sumarização
<code>max_tokens (sumário)</code>	Limite do bloco XML gerado pelo sumarizador
<code>ttd_seconds</code>	Expiração do contexto ativo por inatividade
<code>max_preferences</code>	Corte de preferências por operador injetadas no prompt
<code>max_facts</code>	Corte de fatos do ambiente por operador injetados no prompt
$T_{\text{high}}, T_{\text{low}}$	Limiares superior e inferior da admissão semântica

Fonte: Elaborado pelo autor.

5.2 DESENVOLVIMENTO DO MÓDULO DE MEMÓRIA DE SESSÃO

Esta seção detalha a implementação dos três mecanismos que compõem o módulo de memória de sessão: o pipeline de compactação do histórico, o sumário estruturado gerado pela sumarização e a segmentação dinâmica por tópico. Para cada um, são descritas as decisões de implementação, os problemas encontrados durante o desenvolvimento e as soluções adotadas.

5.2.1 Pipeline de Compactação do Histórico

A primeira versão do pipeline gerenciava o buffer de mensagens manualmente. Com a adoção do *framework* de orquestração, esse gerenciamento migrou para a implementação nativa do *framework*, integrada ao ciclo ReAct, com o limite de mensagens controlado pelo parâmetro `window_size`, reduzindo o código de gerenciamento manual e melhorando a integração com o ciclo do agente. A sumarização permaneceu fora do *framework*, o que exige implementação própria mas mantém controle total sobre os campos e o formato do sumário gerado, sem abrir mão das decisões de projeto descritas na Seção 4.2.2.

A compactação é ativada por dois gatilhos independentes: o buffer atingir o limite de mensagens configurado em `window_size` ou o total de tokens ultrapassar o `threshold_tokens`, o que ocorrer primeiro. A contagem de tokens considera o conteúdo de cada mensagem, mas o modelo acrescenta marcadores de estrutura a cada mensagem, e esses marcadores também consomem tokens. Para compensar, o sistema soma um valor fixo por mensagem à contagem. Como o tokenizador não é exato para todo modelo, aplica-se uma margem de 15%, disparando a compactação um pouco antes do limite nominal para evitar estouros. Essa margem atende diretamente ao RNF-01, que exige que os tokens de entrada não cresçam sem controle em sessões longas.

A sumarização é executada de forma assíncrona, após a resposta ser entregue ao operador. Isso garante que o processamento da compactação não adicione latência ao turno, atendendo ao RNF-02.

5.2.2 Sumário Estruturado de Sessão

A Seção 4.2.2 descreveu a estrutura do bloco XML e os campos que o compõem. Esta subseção detalha como o prompt de sumarização foi construído para produzir esse resultado de forma consistente.

O sumariador recebe três blocos de entrada:

- `<previous_summary>`: o resumo anterior, vazio na primeira compactação;
- `<conversation>`: as mensagens mais antigas da janela, o trecho que será descartado do buffer;
- `<recent_messages>`: as mensagens mais recentes da janela, usadas tanto na sumarização quanto na avaliação de continuidade.

A separação em dois blocos de mensagens foi introduzida após um problema identificado durante o desenvolvimento: quando apenas as mensagens antigas eram sumarizadas, o modelo não enxergava o contexto mais recente da conversa e tendia a

classificar o tópico como contínuo na maioria dos casos, mantendo o mesmo registro indefinidamente. A solução foi incluir as mensagens mais recentes na sumarização também, passando-as num bloco separado para que o modelo avalie a continuidade com base no contexto completo da janela.

O Quadro 11 apresenta a estrutura do prompt de sumarização.

Quadro 11 – Estrutura do prompt de sumarização de sessão.

You are a conversation summarizer. Your task is to produce or update a
 ↳ structured XML summary of a conversation.

If a <previous_summary> is provided, merge it with the new messages: update
 ↳ existing sections, add new information,
 and remove only items that have been explicitly resolved. Never discard
 ↳ unresolved open items.

Output ONLY a single XML block in this exact format:

```
<summary>
  <title>3-5 word noun phrase labeling the main topic</title>
  <continuity>continuous | discontinuous</continuity>
  <intent>The user's primary goal or request in this session</intent>
  <key_facts>Key findings and conclusions - summarize large datasets,
  do not reproduce lists verbatim</key_facts>
  <decisions>Choices made, approaches agreed upon, or conclusions
  ↳ reached</decisions>
</summary>
```

Rules:

- <title>: 3 to 5 words, noun phrase, no verbs. Always reflect the user's
 ↳ CURRENT focus (aligned with <intent>).
 If the user's focus has shifted since the previous summary, update the title -
 ↳ do NOT preserve the old title.
- <continuity>: use ALL available messages (<conversation> and <recent_messages>
 ↳ if provided)
 to evaluate continuity against <previous_summary>:
 - "continuous" - same topic and same goal, conversation is still progressing
 ↳ on the same subject
 - "discontinuous" - topic has shifted in any way: new subject, sub-topic
 ↳ drift, different domain,
 or generic greeting after prior context
 - If no <previous_summary> is provided, always use "discontinuous"
 - When in doubt, prefer "discontinuous"
- Summarize large lists and data payloads into compact descriptions; never
 ↳ reproduce raw data verbatim
- Preserve exact values when they are directly relevant to ongoing context (e.g.
 ↳ a specific ID being investigated)
- Discard greetings, acknowledgements, and pure repetitions
- When updating an existing summary, treat each field as append-only
 unless the new messages explicitly contradict a prior fact.
 If a prior fact is not mentioned in the new messages, preserve it exactly as
 ↳ written.
- Output the <summary> block only - no prose before or after

Fonte: Elaborado pelo autor.

O Quadro 12 apresenta um exemplo de bloco XML gerado pelo sumarizador, ex-

traído da base de desenvolvimento com os identificadores de máquina anonimizados.

Quadro 12 – Exemplo de sumário estruturado gerado pelo sumarizador.

```
<summary>
  <title>CT01 SpeedLoss Analysis</title>
  <continuity>continuous</continuity>
  <intent>The user's goal is to analyze the root causes of SpeedLoss affecting
    ↳ OEE</intent>
  <key_facts>Current OEE is 54,74%, primarily impacted by 365,6 min of
    ↳ SpeedLoss; no specific submotives identified for SpeedLoss in the
    ↳ snapshot; during hours with OEE 0%, the state recorded was primarily
    ↳ "Produzindo" with some instances of "Sem Programacao PCP".</key_facts>
  <decisions>Next step involves a deeper drilldown into operational
    ↳ states/events beyond the current snapshot to identify root causes of
    ↳ SpeedLoss.</decisions>
</summary>
```

Fonte: Base de desenvolvimento do sistema LivIA, com identificadores anonimizados.

O exemplo mostra o sumarizador operando como esperado. O campo *continuity* marcado como *continuous* indica que o tópico não mudou entre compactações. O *key_facts* condensou dados quantitativos da sessão em uma frase compacta, permitindo que o agente retome o contexto sem reprocessar o histórico bruto. Já o *decisions* registrou o próximo passo definido pelo operador, disponível na próxima sessão como parte do [SESSION CONTEXT].

Durante o desenvolvimento, um campo adicional foi tentado e descartado. Inicialmente, o sumário estruturado incluía o campo `<open_items>` para rastrear itens não resolvidos entre compactações. O problema central era que o fechamento de um item só era refletido na compactação seguinte, nunca na atual. Por conta disso, o agente carregava itens em aberto que já haviam sido resolvidos, sem conseguir fechá-los no mesmo ciclo em que foram concluídos. Esse acúmulo levou a um comportamento indesejado: quando ocorria algum erro interno do agente ou quando ele se perdia no contexto, em vez de responder à pergunta original, retomava os itens listados em `<open_items>` e os tratava como parte do contexto atual, produzindo respostas fora do assunto que pareciam com alucinações. Por isso, o campo foi removido e a continuidade de itens pendentes passou a ser absorvida pelo *intent* e pelos *key_facts*.

5.2.3 Controle de Continuidade de Tópico e Persistência do Registro de Sessão

O mecanismo de *topic_sequence* descrito na Seção 4.2.3 exige uma função de escrita que decida, a cada compactação, se o registro existente deve ser atualizado ou se um novo deve ser criado. Esse mecanismo aplica duas regras em ordem de prioridade antes de avaliar o campo *continuity* retornado pelo sumarizador.

A primeira regra é a de anti-fragmentação. Se o buffer tiver menos mensagens que o *window_size* configurado, o sistema atualiza o registro existente independente do valor de *continuity*. Essa situação ocorre quando o buffer ultrapassa o limite de tokens com poucas mensagens. Sem essa regra, o sistema criaria um novo registro com um único turno, fragmentando o histórico de forma desnecessária.

A segunda regra é a de virada de dia. Antes de avaliar continuidade, o sistema verifica a data do registro ativo. Se o registro pertence a um dia anterior, um novo registro é criado com o *topic_sequence* incrementado, independente da classificação do sumário e mesmo que seja o mesmo assunto. Isso garante que o histórico de um dia de trabalho nunca seja sobrescrito pelo dia seguinte, e conecta à lógica de recuperação descrita na Seção 5.2.4, em que cada dia representa uma sessão de trabalho distinta.

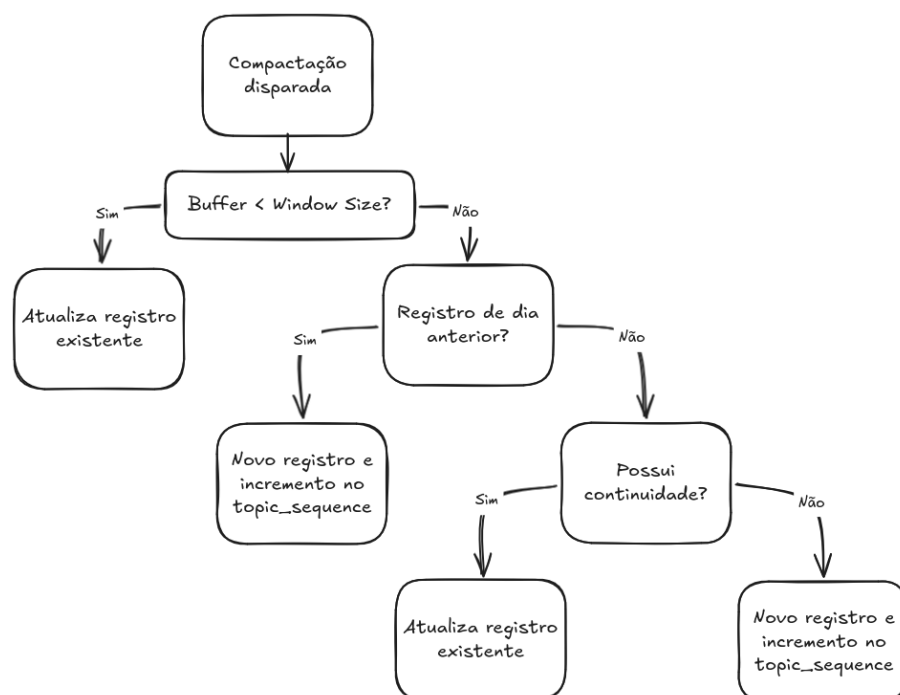
As duas regras seguintes são relacionadas com o resultado do campo *continuity*. Se o valor for *continuous*, o registro atual é atualizado. Se for *discontinuous*, um novo registro é criado com *topic_sequence* incrementado.

A classificação de continuidade passou por uma revisão durante o desenvolvimento. Inicialmente, o sistema adotava três classificações seguindo o modelo do MemBox, sendo elas: *continuous*, *partial_shift* e *discontinuous*. O *partial_shift* sinalizava uma mudança parcial de assunto, sem romper completamente com o contexto anterior. Porém, na prática, esse estado criou um problema. Conversas que derivavam gradualmente de tópico nunca chegavam a disparar o *discontinuous*. A cada compactação, o LLM classificava a mudança como *partial_shift* e o sistema mantinha o mesmo resumo, acumulando contextos distintos em um único bloco. Quando a conversa finalmente estava completamente diferente do início, não havia registro separado do novo tópico.

O *partial_shift* então foi removido e a classificação passou a ser apenas binária. Com isso, a regra ficou direta, ou a conversa continua no mesmo assunto, ou não. Para guiar o LLM nessa decisão, o *prompt* estabelece que em caso de dúvida o resultado deve ser *discontinuous*. Adicionar um novo registro é menos custoso do que perder uma informação importante que seja sobrescrita por um tópico atual.

A Figura 19 apresenta o fluxo completo de decisão.

Figura 19 – Fluxo de decisão para persistência do registro de sessão.



Fonte: Elaborado pelo autor.

O Quadro 13 apresenta os campos da tabela *session_snapshots*, responsável por armazenar os resumos gerados pelo sumariizador.

Quadro 13 – Campos da tabela *session_snapshots*.

Campo	Descrição
session_id	Identifica a sessão
user_id	Identifica o operador
company_id	Identifica a empresa
date_snapshot	Data do registro
topic_sequence	Número do tópico dentro da sessão
title	Título gerado pelo sumariizador
summary	Sumário gerado pelo sumariizador
messages	Mensagens do buffer

Fonte: Elaborado pelo autor.

Foi implementada nas configurações do banco uma restrição de unicidade composta por (*company_id*, *user_id*, *session_id*, *topic_sequence*) que é responsável por cumprir dois papéis: permite que o registro existente seja atualizado sem duplicar quando o tópico é contínuo (a cada compactação o sumário acumula novas informações e, sem essa restrição, cada atualização criaria um registro novo em vez de

sobrescrever o anterior), e garante que registros de empresas distintas nunca colidam no banco, reforçando o RF-08 e o RNF-05. A inclusão de *company_id* na chave tornou explícita no banco uma garantia que antes dependia apenas do código, pois toda consulta filtrava por *company_id* manualmente, mas nada impedia que um erro de implementação omitisse esse filtro. Com *company_id* na chave, o banco previne qualquer colisão entre empresas independente do que o código faça. A coluna de data sustenta a regra de virada de dia e o filtro por data na ferramenta de busca de histórico.

5.2.4 Recuperação de Sessão

O *session_id* de cada conversa é composto pela concatenação de *user_id* e *company_id*, formando um escopo que isola as mensagens por operador e por empresa em todas as operações de leitura e escrita. Esse escopo é aplicado consistentemente nas chaves do cache em memória, nos predicados de consulta ao banco de dados relacional e no escopo da ferramenta de busca de histórico.

O contexto ativo de uma sessão no cache em memória possui um prazo de expiração, que é renovado a cada escrita. Quando o cache esvazia por inatividade, reinicialização ou primeira interação após um longo intervalo, o sistema executa a recuperação do contexto em dois níveis. Primeiro, verifica se o cache possui a chave de sessão. Se não encontrar, consulta o banco de dados relacional buscando o registro mais recente do operador. A partir do registro recuperado, aplica a regra do mesmo dia: se a data do registro for a data atual, o sistema recupera o contexto da sessão para o cache, restabelecendo o buffer de mensagens e o resumo estruturado. Se o registro for de um dia anterior, a sessão começa vazia. O histórico de um dia anterior não retorna ao contexto ativo, mas permanece acessível sob demanda pela ferramenta *search_session_history*, descrita na Seção 5.3.5.

Essa regra conecta ao mecanismo de virada de dia descrito anteriormente e ao conceito de *context rot*, a degradação do agente por acúmulo de contexto desatualizado, discutido no Capítulo 3. Iniciar o turno com um buffer limpo é preferível a carregar contexto que pode ser irrelevante ou contraditório com a situação atual da operação.

A atualização do resumo em cache é escrita apenas no cache em memória, sem persistir no banco de dados relacional durante o turno. A escrita definitiva ocorre exclusivamente ao final da compactação. Gravar no banco a cada turno poderia sobrescrever um registro anterior sem passar pelas quatro regras de decisão, corrompendo a lógica do *topic_sequence*.

5.3 DESENVOLVIMENTO DO MÓDULO DE MEMÓRIA DE LONGO PRAZO

Esta seção detalha a implementação do módulo de memória de longo prazo: o extrator unificado que identifica e roteia memórias por tipo, as estruturas de dados e

tabelas que as armazenam, a política de admissão semântica que controla o que entra no banco, e os mecanismos de recuperação passiva e ativa que disponibilizam esse conhecimento ao agente.

5.3.1 Extração Unificada de Memória

A implementação inicial separava a extração de preferências e a extração de fatos do ambiente em dois módulos distintos, cada um com seu próprio regex de gating e prompt. Com a evolução do sistema, os dois módulos foram consolidados em um único arquivo que concentra os dois regexes, um prompt unificado e a função que é responsável por separar e rotear a saída do LLM.

O operador lógico OR entre os dois padrões de detecção forma o gate de memória: se qualquer um disparar, uma única chamada ao LLM de extração é feita, independente de quantos subsistemas estejam ativos. O custo máximo por turno é uma chamada de extração, mesmo com preferências e fatos habilitados simultaneamente.

A saída do LLM é separada pelo campo *kind* e cada tipo é salvo na tabela correspondente no banco: preferências em `user_preferences` e fatos do ambiente em `environment_facts`.

O Quadro 14 apresenta o prompt de extração.

Quadro 14 – Prompt do extrator unificado de memória de longo prazo.

You extract durable long-term memory from a conversation turn. Each memory item
 ↪ is ONE of two kinds:

- "preference": HOW the user wants to interact (durable across sessions), e.g. communication style ("seja conciso"), language ("responda em português"), formatting ("use duas casas decimais para OEE").
- "environment_fact": a durable fact about the operator's working environment - who they are and where/when they operate, e.g. sector ("trabalha no setor ↪ {sector}"), shift ("trabalha no turno {shift}"), role ("é responsável pela linha {line}"), equipment they routinely operate ("opera a {CT}").

Output a JSON array. Each item must have:

- kind: "preference" or "environment_fact"
- exchange_core: the essence, stated concisely (short, max 80 chars)
- specific_context: when/where it applies (concise sentence; null if it always ↪ applies)
- room: a short lowercase category. For preferences use "communication" | ↪ "language" | "format".
- For environment facts use a fitting label, e.g. "sector" | "shift" | "role" | ↪ "equipment".

Rules:

- CRITICAL: Output ALWAYS in the SAME language as the user's message. If the ↪ user wrote in Portuguese, output in Portuguese. If in English, output in English. Keep the ↪ original wording.
- Extract an item only when it is durable - true across sessions, not just for ↪ this request.
- If EXISTING_MEMORY is provided, extract only genuinely new items; skip any ↪ that an existing item already states, even if phrased differently or in another language.
- When the turn carries no durable memory, return [].

Treat transient operational queries ("qual o OEE da {CT} hoje?", "liste as ↪ paradas de ontem", "compare {CT} com {CT2}") as requests, not memory - they describe what the ↪ operator wants now, not a durable truth or preference.

Fonte: Elaborado pelo autor.

A escolha do JSON para o extrator contrasta com o XML adotado no sumariizador. Os campos extraídos são curtos e bem delimitados, como *exchange_core* e *specific_context*, o que torna o JSON mais adequado que o XML nesse caso. Como o modelo nem sempre retorna JSON puro, às vezes envolvendo a saída em um bloco de código ou adicionando texto ao redor, o *parser* extrai a lista corretamente mesmo

nesses casos.

Um problema identificado durante o desenvolvimento foi a contaminação da base de memórias por consultas pontuais. O extrator interpretava perguntas como “qual o OEE do CT01 hoje?” como fatos do ambiente, inserindo registros que não se sustentam entre sessões. A instrução que distingue fatos e preferências do operador de consultas do dia a dia, com exemplos explícitos no prompt, foi adicionada após esse comportamento ser observado e motivou o refinamento descrito na Seção 4.3.2.

5.3.2 Preferências do Usuário e Fatos do Ambiente

As tabelas *environment_facts* e *user_preferences* compartilham o mesmo esquema. O Quadro 15 apresenta os campos de ambas.

Quadro 15 – Campos das tabelas *environment_facts* e *user_preferences*.

Campo	Descrição
<code>session_id</code>	Identifica a sessão em que o registro foi extraído
<code>user_id</code>	Identifica o operador
<code>company_id</code>	Identifica a empresa
<code>date_fact</code>	Data do registro
<code>exchange_core</code>	Essência da preferência ou fato
<code>specific_context</code>	Contexto de aplicação, nulo quando sempre válido
<code>room</code>	Categoria do registro
<code>source</code>	Origem da extração
<code>embedding</code>	Vetor de 1.536 dimensões utilizado na admissão semântica

Fonte: Elaborado pelo autor.

O Quadro 16 apresenta o esquema relacional compartilhado pelas duas tabelas.

Quadro 16 – Esquema relacional compartilhado pelas tabelas *environment_facts* e *user_preferences*.

```

id                UUID PRIMARY KEY,
session_id        TEXT NOT NULL,
user_id           TEXT NOT NULL,
company_id        TEXT NOT NULL,
date_fact         DATE NOT NULL DEFAULT CURRENT_DATE,
exchange_core     TEXT NOT NULL,
specific_context  TEXT,
room              TEXT NOT NULL,
source            TEXT NOT NULL CHECK (source IN ('heuristic_turn',
↪ 'compaction')),
embedding         vector(1536),
created_at        TIMESTAMPTZ NOT NULL DEFAULT now(),
updated_at        TIMESTAMPTZ NOT NULL DEFAULT now()

```

Fonte: Elaborado pelo autor.

A versão inicial usava uma restrição de unicidade sobre (*user_id*, *company_id*, *session_id*, *exchange_core*), de modo que quando um registro com o mesmo texto chegava, o banco sobrescrevia o anterior via `ON CONFLICT`. Essa abordagem só detectava registros textualmente idênticos e ignorava variações (como “seja conciso” e “responda de forma curta”) ou itens complementares (como “use 2 casas decimais para o OEE” e “mostre o tempo de parada em minutos”), que são distintos e devem coexistir. Com a introdução dos embeddings, a restrição foi removida e a deduplicação passou para a política de admissão semântica, descrita na Seção 5.3.3, que cobre casos que a igualdade textual não detecta.

5.3.3 Controle de Admissão de Memória

Antes de persistir um candidato, preferência ou fato do ambiente, o sistema executa a política de admissão descrita na Seção 4.3.4. A implementação começa pela vetorização: o texto é incorporado com o modelo de *embeddings* adotado no sistema, evitando a necessidade de um segundo modelo de *embedding*. O mesmo formato é usado na injeção no *system prompt* e na calibração dos limiares, garantindo que o que é comparado em desenvolvimento corresponde ao que é comparado em produção.

O vetor é persistido junto ao registro em uma coluna de embedding indexada por um índice HNSW, um algoritmo de busca de vizinhos mais próximos baseado em grafos. A métrica usada é a de cosseno. Dessa forma, a comparação na próxima admissão busca os vizinhos mais próximos diretamente no banco, sem precisar reincorporar os registros já armazenados.

A política de admissão classifica o candidato em três zonas com base na distância de cosseno d em relação ao vizinho mais próximo já armazenado, conforme o Quadro 17.

Quadro 17 – Zonas de decisão da política de admissão A-MAC.

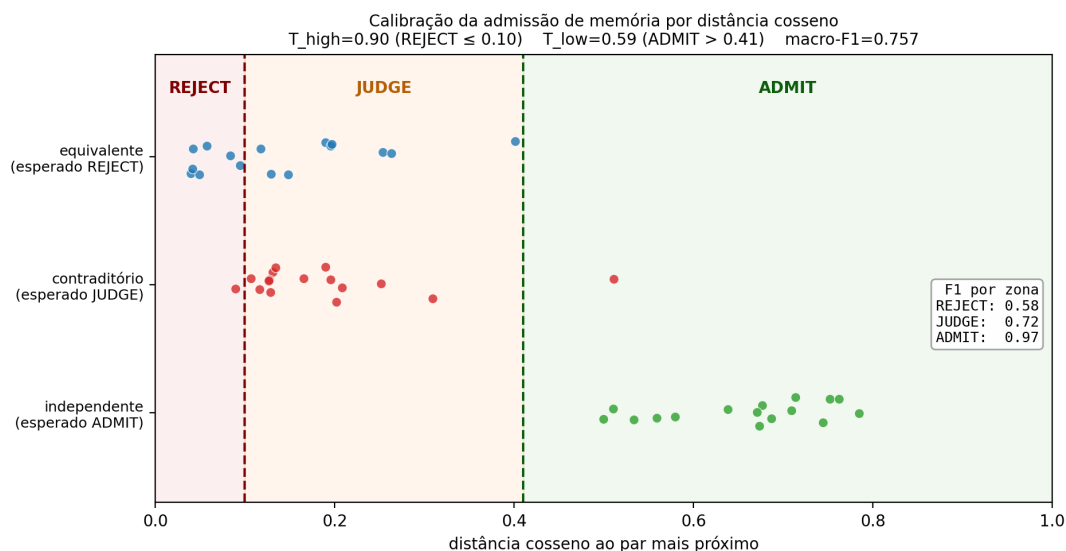
Distância d	Decisão	Ação
$d \leq 0,10$	REJECT	Descarta o candidato (duplicata semântica)
$0,10 < d \leq 0,41$	Árbitro LLM	Contradiz? UPDATE (remove top-1, insere novo); senão ADMIT
$d > 0,41$	ADMIT	Insere como registro independente

Fonte: Elaborado pelo autor.

Os limiares $T_{\text{high}} = 0,90$ e $T_{\text{low}} = 0,59$ correspondem a distâncias de $1 - T_{\text{high}} = 0,10$ e $1 - T_{\text{low}} = 0,41$. O T_{high} alto é intencional: variações e contradições ficam à mesma distância do vizinho no espaço de embedding, o que significa que o REJECT automático só pode descartar o quase-idêntico. Qualquer candidato que uma contradição alcance precisa ir ao árbitro, pois a distância sozinha não consegue separar os dois casos.

Os limiares foram calibrados com um script desenvolvido pelo autor a partir de 16 triplas em português do Brasil. Cada tripla parte de uma memória já existente no banco, como “responda em português”, e inclui três candidatos. Uma variante que diz a mesma coisa com palavras diferentes, como “responda sempre em português”, esperada na zona de rejeição. Uma variante que contradiz a memória original, como “responda em inglês”, esperada na zona do árbitro LLM. E um candidato sobre outro assunto, como “use 2 casas decimais para o OEE”, esperado na zona de admissão direta. Todo esse conjunto resulta no total de 48 pares. O script opera em dois modos: leitura de triplas prontas de um arquivo JSON ou geração via LLM. O mesmo modelo de embedding e o mesmo formato de texto de preferências e fatos do sistema em produção são usados na calibração, garantindo que os limiares obtidos valem para o ambiente real. A Figura 20 mostra a distribuição das distâncias de cosseno por classe, com as três zonas sombreadas e os limiares de corte marcados.

Figura 20 – Distribuição das distâncias de cosseno por classe e zonas de admissão resultantes da calibração.



O F1 é uma métrica que mede a qualidade de um classificador em atribuir corretamente os pares ao rótulo de uma zona: vale 1 quando todos os pares são classificados corretamente e 0 quando nenhum é. O macro-F1 é a média do F1 de cada zona, tratando todas com igual peso. O resultado de 0,757 é a média das três zonas: ADMIT (0,97), REJECT (0,58) e árbitro LLM (0,72), e reflete o comportamento esperado de cada uma. A zona ADMIT obteve F1 de 0,97: candidatos independentes, como “mostre o tempo de parada em minutos” diante de uma memória sobre preferência de idioma, se separam com clareza no espaço de *embedding* e quase nunca chegam ao árbitro. A zona REJECT obteve F1 de 0,58 por conta de escolhermos um $T_{high} = 0,90$ que é conservador por design. O motivo é que apenas parte dos equivalentes é descartada automaticamente e o restante segue para o árbitro. Esse é o comportamento seguro desejado. O teto do macro-F1 é imposto pela sobreposição na zona do árbitro LLM: uma memória como “responda em português” e sua contradição “responda em inglês” ficam à mesma distância de cosseno no espaço de *embedding*, porque os dois textos compartilham estrutura e vocabulário. A distância sozinha não distingue variação de contradição, e é exatamente esse limite que justifica o árbitro LLM nessa região.

5.3.4 Corte de Memória e Controle de Compactação

O compactador implementa um mecanismo de proteção contra compactações que possuem pouca redução de tokens. Se duas compactações consecutivas economizaram menos de 10% dos tokens do *buffer*, a próxima é pulada. Quando o *buffer* já está razoavelmente compacto, sumarizar novamente gera pouco ganho e consome tokens de chamadas ao LLM e tempo desnecessários. Como a extração de memória

de longo prazo é disparada junto à sumarização, o mecanismo também evita que o extrator rode sem ganho real, atendendo ao RNF-02 e ao RNF-03.

O número máximo de memórias injetadas no prompt é controlado pelos parâmetros *max_preferences* e *max_facts*, com padrão de 50 registros cada, aplicados pela cláusula `LIMIT` nas consultas ao banco. Sem esse limite, os blocos `[USER PREFERENCES]` e `[ENVIRONMENT FACTS]` cresceriam proporcionalmente ao número de memórias do operador, inflando o *system prompt* e degradando o RNF-01 e o RNF-03.

5.3.5 Recuperação Híbrida

A busca de histórico usa a extensão de trigramas do banco de dados relacional adotado, que indexa texto por trigramas, sequências de três caracteres sobrepostos, e calcula a similaridade entre duas strings como a proporção de trigramas em comum. Ela foi escolhida porque os títulos das sessões são frases curtas, geralmente de três a cinco palavras, para as quais tolera variações de espaçamento como “CM 026” e “CM026” sem custo de *embedding*. Essa proporção é um valor entre 0 e 1, onde 1 indica strings idênticas e 0 indica nenhum trigrama em comum. Apenas títulos com proporção acima de 0,35 são retornados, valor um pouco acima do limiar padrão de 0,3 adotado pela função de similaridade de trigramas do banco de dados relacional utilizado, o que torna a correspondência ligeiramente mais estrita. Para frases curtas de três a cinco palavras o número absoluto de trigramas é pequeno, de modo que 0,35 já exige uma sobreposição léxica considerável entre o termo buscado e o título. A busca vetorial fica reservada para a admissão semântica, onde a distância entre textos completos importa.

A ferramenta suporta quatro combinações de filtro: texto com data, texto, data e nenhum filtro. Nesse último caso, retorna os registros mais recentes por padrão. Datas relativas como “ontem” ou “semana passada” são convertidas pelo próprio modelo de linguagem para o formato YYYY-MM-DD antes da chamada, usando a data atual presente no *system prompt*. O resultado é limitado a cinco registros e ordenado do mais recente para o mais antigo, já que, como descrito na Seção 4.3.3, referências a conversas passadas quase sempre carregam uma âncora temporal.

5.4 AVALIAÇÃO DE DESEMPENHO

Esta seção avalia em que medida o sistema implementado atende os requisitos definidos no Capítulo 3 e apresenta os resultados de desempenho obtidos. A Seção 5.4.1 examina cada requisito funcional e não funcional à luz dos mecanismos implementados e dos testes realizados. A Seção 5.4.3 discute a eficiência de contexto, custo e latência a partir de uma comparação entre configurações de memória.

5.4.1 Avaliação por Requisitos Funcionais e Não Funcionais

A estratégia de avaliação adotada foca em verificar se o sistema age corretamente, com base em comportamento determinístico. Avaliar se as respostas ficam semanticamente melhores exigiria um árbitro LLM com rubrica avaliadora, o que introduziria variabilidade e ruído na própria medição. Os requisitos funcionais de persistência e isolamento são verificados por duas camadas de testes, descritas a seguir, enquanto os requisitos não funcionais de eficiência, custo e latência são medidos na Seção 5.4.3.

Os requisitos são verificados por testes determinísticos em dois níveis. A primeira camada usa testes unitários, que verificam cada função de forma isolada, e testes de integração, que verificam a interação entre componentes como extrator, banco relacional e banco vetorial. Esses testes são executados automaticamente no Integração Contínua (*Continuous Integration*) (CI), *pipeline* que roda a suíte a cada nova alteração no repositório. A segunda camada usa testes *end-to-end* ao vivo, com agente e banco reais. Cada teste grava um token único por execução, aguarda a escrita assíncrona aparecer no banco e verifica isolamento, recuperação entre sessões e atualização por contradição.

O caso mais relevante é o RF-07, que demonstra a continuidade entre sessões. Um fato estabelecido na sessão A é recuperado corretamente na sessão B, que inicia com buffer vazio. Como a sessão B não tem acesso ao histórico da sessão A, a única origem possível da resposta correta é o banco de memória de longo prazo, tornando a verificação completamente determinística. Esse caso é ilustrado na Seção 5.4.2. O Quadro 18 consolida o resultado por requisito, indicando o mecanismo de atendimento correspondente.

Os limiares da política de admissão foram calibrados na Seção 5.3.3, resultando em $T_{\text{high}} = 0,90$ e $T_{\text{low}} = 0,59$, com macro-F1 de 0,757.

O Quadro 18 consolida o resultado da avaliação para cada requisito.

Quadro 18 – Avaliação dos requisitos funcionais e não funcionais.

Código	Mecanismo de atendimento	Resultado
RF-01	Buffer nativo do framework de orquestração com janela deslizante de N mensagens	Atendido
RF-02	Compactor acionado por limitador de tokens. Gera sumário estruturado e reduz o buffer	Atendido
RF-03	Segmentação por tópico com quatro regras de decisão	Atendido
RF-04	Sumário em XML com campos próprios persistido por sessão	Atendido
RF-05	Extrator unificado acionado por <i>regex</i> após cada turno e na compactação	Atendido
RF-06	Tabelas <i>user_preferences</i> e <i>environment_facts</i> indexadas por <i>user_id</i> e <i>company_id</i>	Atendido
RF-07	Injeção de preferências e fatos antes do primeiro turno de cada nova sessão	Atendido
RF-08	Toda consulta ao banco filtra por <i>user_id</i> e <i>company_id</i> , sem caminho de acesso cruzado	Atendido
RF-09	Árbitro LLM decide contradições, removendo o registro antigo e inserindo o novo	Atendido
RF-10	Candidato com distância de cosseno $d \leq 0,10$ é descartado pela zona de rejeição	Atendido
RNF-01	Compactação mantém o buffer limitado, corte de <i>max_preferences</i> e <i>max_facts</i> limita injeção	Atendido
RNF-02	Operações de memória executadas de forma assíncrona, fora do caminho síncrono de resposta	Atendido
RNF-03	Compactação limita o crescimento de tokens por turno; <i>cache</i> de prefixo amortiza o custo do histórico	Parcial
RNF-04	Fato injetado é idêntico ao armazenado, política de admissão semântica impede duplicatas e versões obsoletas	Parcial
RNF-05	Idêntico ao RF-08, índice exclusivo por <i>user_id</i> e <i>company_id</i> em todas as tabelas	Atendido
RNF-06	Zona de rejeição da política de admissão semântica garante que paráfrases não geram registros duplicados	Atendido
RNF-07	Extração e persistência de longo prazo executadas de forma assíncrona, sem bloquear a resposta ao operador	Atendido

Fonte: Elaborado pelo autor.

O RNF-04 exige que a taxa de alucinação não aumente com a memória ativa. Por projeto, os fatos injetados no *prompt* são idênticos aos armazenados, sem transformação no caminho entre banco e contexto, e a política de admissão semântica impede que fatos obsoletos ou contraditórios permaneçam ativos, por meio dos mecanismos descritos em RF-09 e RF-10. Isso controla a qualidade do conteúdo injetado, mas não elimina o risco de alucinação decorrente do aumento de tokens no contexto, efeito discutido na Capítulo 2. A verificação completa do requisito exigiria um árbitro LLM

com rubrica avaliadora e é deixada como trabalho futuro na Capítulo 6. Por isso, o requisito é classificado como parcialmente atendido.

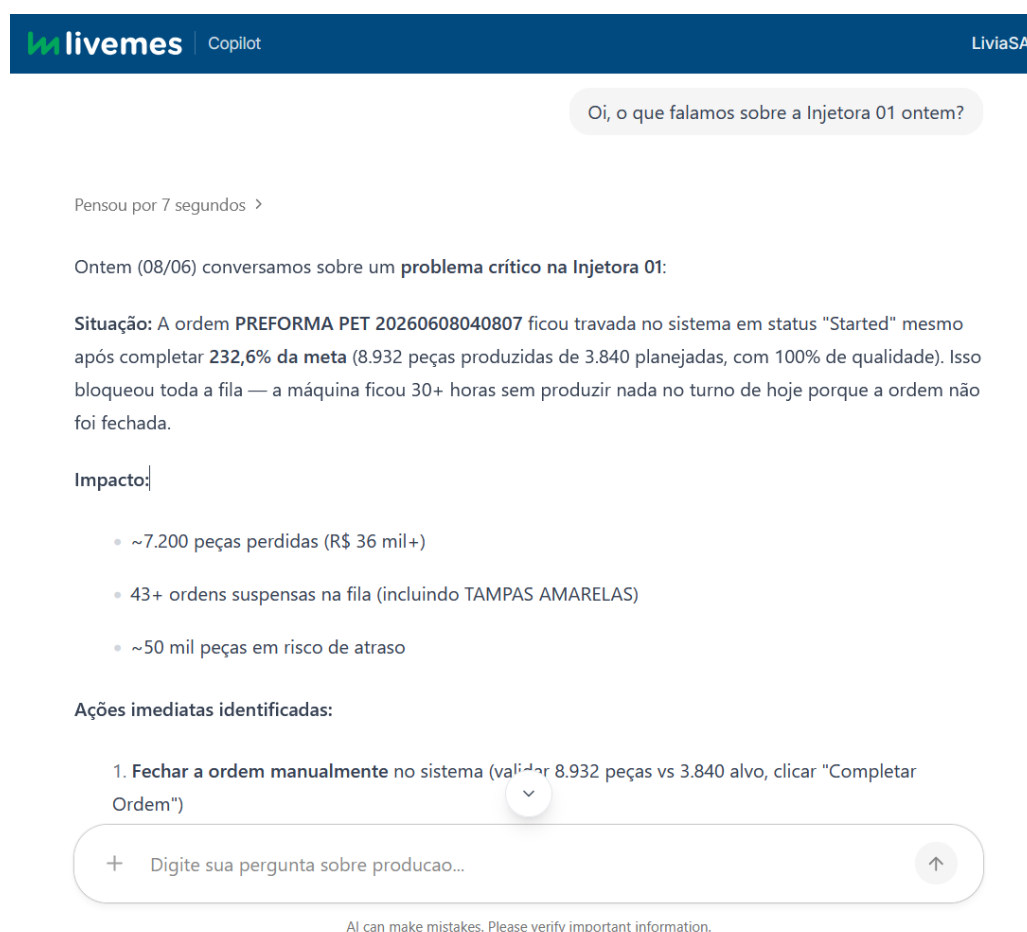
5.4.2 Exemplos de Operação

Esta subseção apresenta dois exemplos concretos que ilustram o funcionamento do sistema em situações representativas de uso real.

5.4.2.1 Recuperação de Histórico de Sessão

O primeiro exemplo demonstra a recuperação ativa de histórico via ferramenta *search_session_history*, descrita na Seção 5.3.5. Para o teste, foi simulada uma conversa entre o operador e o agente até que o sumário estruturado fosse gerado e persistido. Em seguida, a data do registro foi alterada para o dia anterior e o cache em memória foi limpo, simulando uma nova sessão sem continuidade direta. Com o histórico da sessão anterior acessível apenas pelo banco, o operador perguntou sobre a conversa passada e o agente retornou o resultado correto, como mostra a Figura 21.

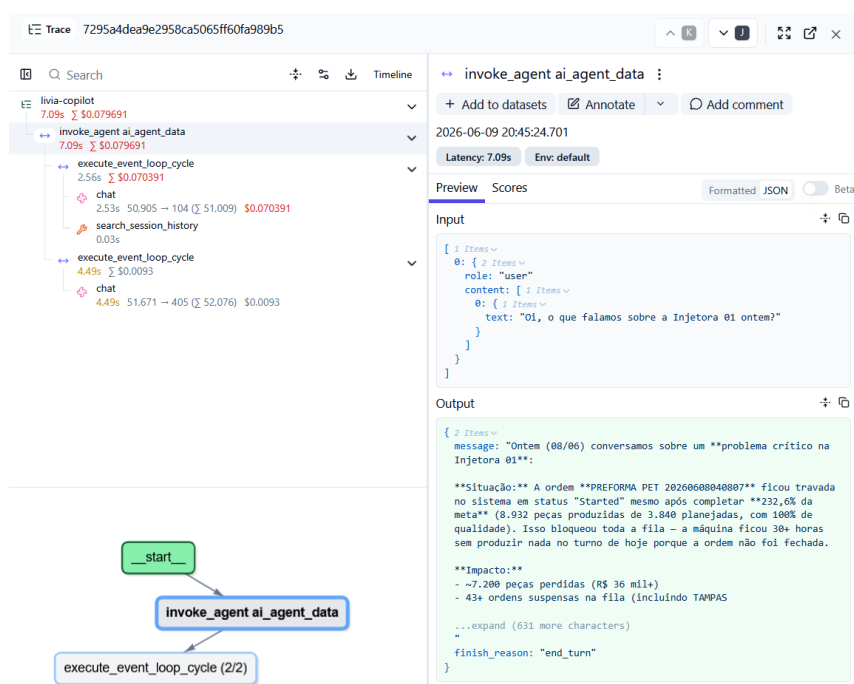
Figura 21 – Operador pergunta sobre conversa anterior e o agente recupera o resumo correto a partir do banco.



Fonte: Elaborado pelo autor.

A Figura 22 mostra o trace completo da interação no Langfuse, ferramenta de observabilidade para aplicações LLM que registra cada chamada ao modelo com seu *input*, *output*, latência, tokens e custo, permitindo rastrear todo o caminho da requisição.

Figura 22 – Trace completo no Langfuse com *input* e *output* da interação.



Fonte: Elaborado pelo autor.

A Figura 23 detalha a chamada à ferramenta, mostrando os parâmetros enviados pelo agente e o conteúdo retornado pelo banco, incluindo o número de registros limitado pelo parâmetro *top_k*.

Figura 23 – Chamada à ferramenta *search_session_history*: parâmetros enviados pelo agente e registros retornados pelo banco com *top_k*.

The screenshot displays the interface for the `search_session_history` tool. At the top, there are buttons for '+ Add to datasets', 'Annotate', and a dropdown menu with 'Playground'. Below these is an 'Add comment' button. The timestamp '2026-06-09 20:45:27.279' and 'Latency: 0.03s' are shown. The 'Preview' tab is active, displaying a JSON query:

```
[{"role": "tool", "content": {"query": "Injetora 01", "top_k": 5}, "id": "tooluse_PAtaBlyUu1vEkFH3jsHioZ"}]
```

. The 'Output' tab shows a detailed response:

```
{ "message": [{"text": "[Hist\u00f3rico relevante encontrado: 1 resultado(s)]"}], "id": "tooluse_PAtaBlyUu1vEkFH3jsHioZ" }
```

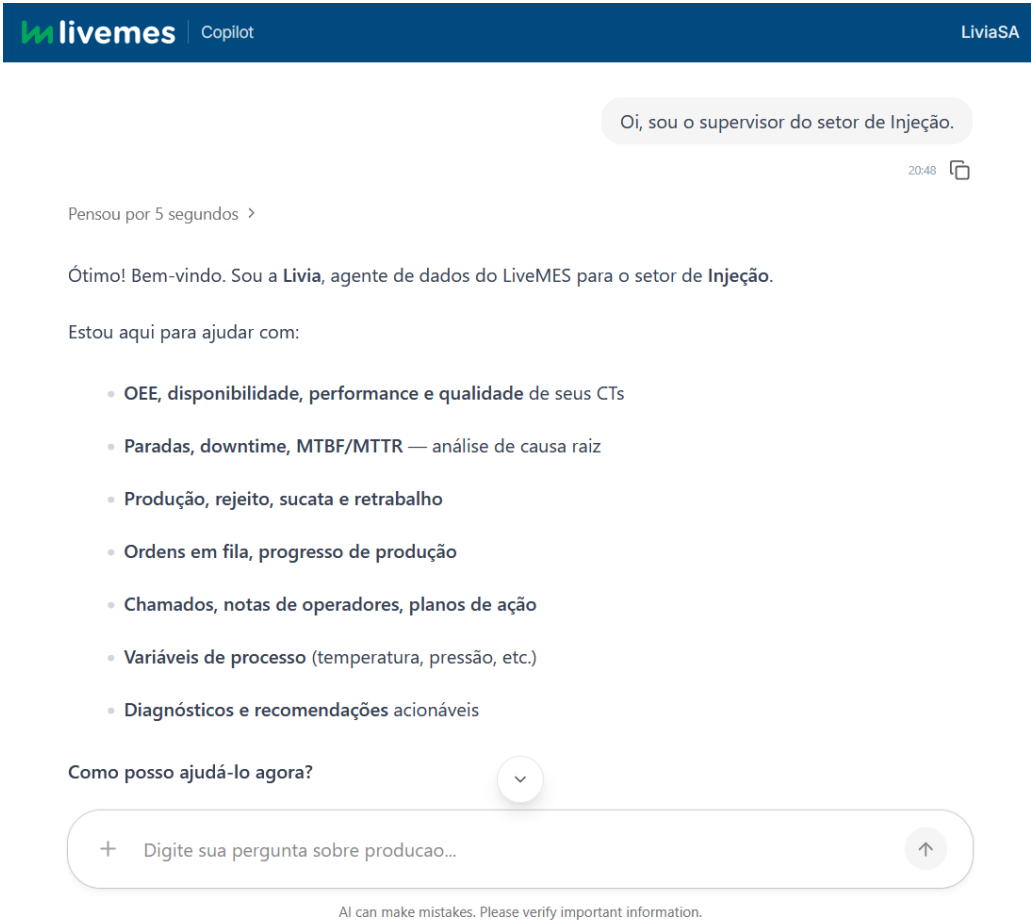
. The output text includes a summary of a production blockage for 'Injetora 01' and a request to expand the text.

Fonte: Elaborado pelo autor.

5.4.2.2 Extração e Injeção de Fatos do Ambiente

O segundo exemplo demonstra o ciclo completo de extração e recuperação de fatos do ambiente. O operador informa ao agente que é supervisor do setor de Injeção, frase que ativa o padrão de detecção por *regex* do extrator, como mostra a Figura 24.

Figura 24 – Operador informa papel e setor ao agente, ativando o padrão de detecção do extrator.



Fonte: Elaborado pelo autor.

A Figura 25 mostra os dois traces gerados: o da mensagem enviada pelo operador e, acima, o do LLM extrator acionado de forma assíncrona após a resposta.

Figura 25 – Dois traces no Langfuse: a mensagem do operador e o LLM extrator acionado de forma assíncrona.

☐	Timestamp ▼	Name	Input	Output
☐	☆ 2026-06-09 20:48:22	livia-copilot	"Oi, sou o supervisor do setor de Injeção."	""json [{ "kind": "environment_fact", "exchange_core": "Supervisor do seto...
☐	☆ 2026-06-09 20:48:16	livia-copilot	"([{"text": "\u201cOi, sou o supervisor do setor de Inje\u00e7\u00e3o.\u201c"])"	("message": "\u201c\u00d3timo! Bem-vindo. Sou a **Livia**, agente de dados do LiveM...

Fonte: Elaborado pelo autor.

As figuras Figura 26 e Figura 27, apresentadas lado a lado, mostram, \u00e0 esquerda, o *system prompt* com o contexto de ambiente usado como entrada e, \u00e0 direita, a sa\u00edda do extrator. A partir de uma \u00fanica frase do operador, o LLM extrator percebe dois fatos distintos, setor de Inje\u00e7\u00e3o e fun\u00e7\u00e3o de supervisor, e os persiste como dois registros independentes no banco.

Figura 26 – *System prompt* do extrator com o contexto de ambiente como entrada.

```
Input
[ 2 Items
0: { 2 Items
  role: "system"
  content: "You extract durable long-term memory from a conversation turn. Each memory item is ONE of two kinds:
    - 'preference': HOW the user wants to interact (durable across sessions), e.g. communication style ('seja conciso'), language ('responda em português'), formatting ('use duas casas decimais para OEE').
    - 'environment_fact': a durable fact about the operator's working environment - who they are and where/when they operate, e.g. sector ('trabalha no setor Compressoras'), shift ('trabalha no ...expand (1256 more characters)
  }
1: { 2 Items
  role: "user"
  content: "Oi, sou o supervisor do setor de Injeção."
}
]
```

Fonte: Elaborado pelo autor.

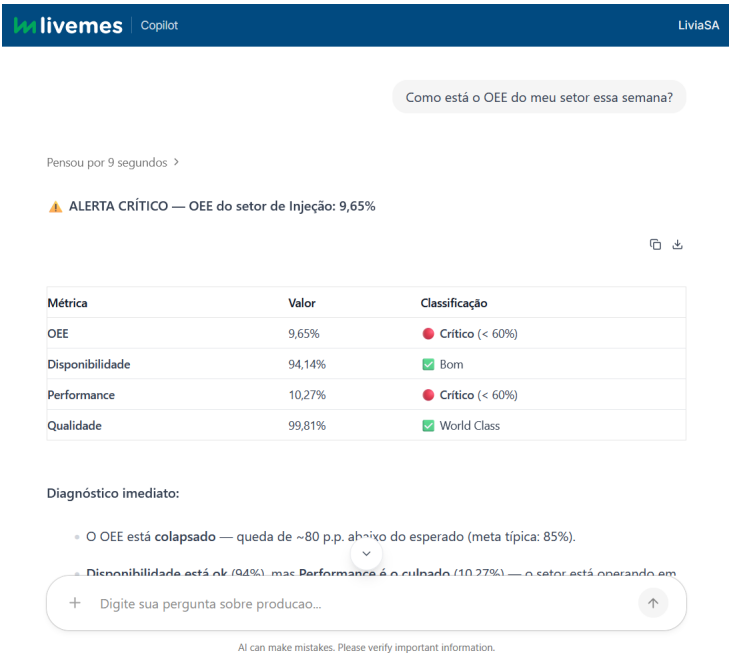
Figura 27 – Saída do extrator: setor de Injeção e função de supervisor salvos como dois registros distintos.

```
Output
```json
[
 {
 "kind": "environment_fact",
 "exchange_core": "Supervisor do setor de Injeção",
 "specific_context": null,
 "room": "role"
 },
 {
 "kind": "environment_fact",
 "exchange_core": "Trabalha no setor de Injeção",
 "specific_context": null,
 "room": "sector"
 }
]
```

Fonte: Elaborado pelo autor.

Em uma nova sessão com cache limpo, o operador perguntou apenas sobre o OEE do seu setor, sem informar qual era seu setor. O agente respondeu com o OEE da Injetora, como mostra a Figura 28, demonstrando que o fato persistido foi recuperado e injetado corretamente no contexto.

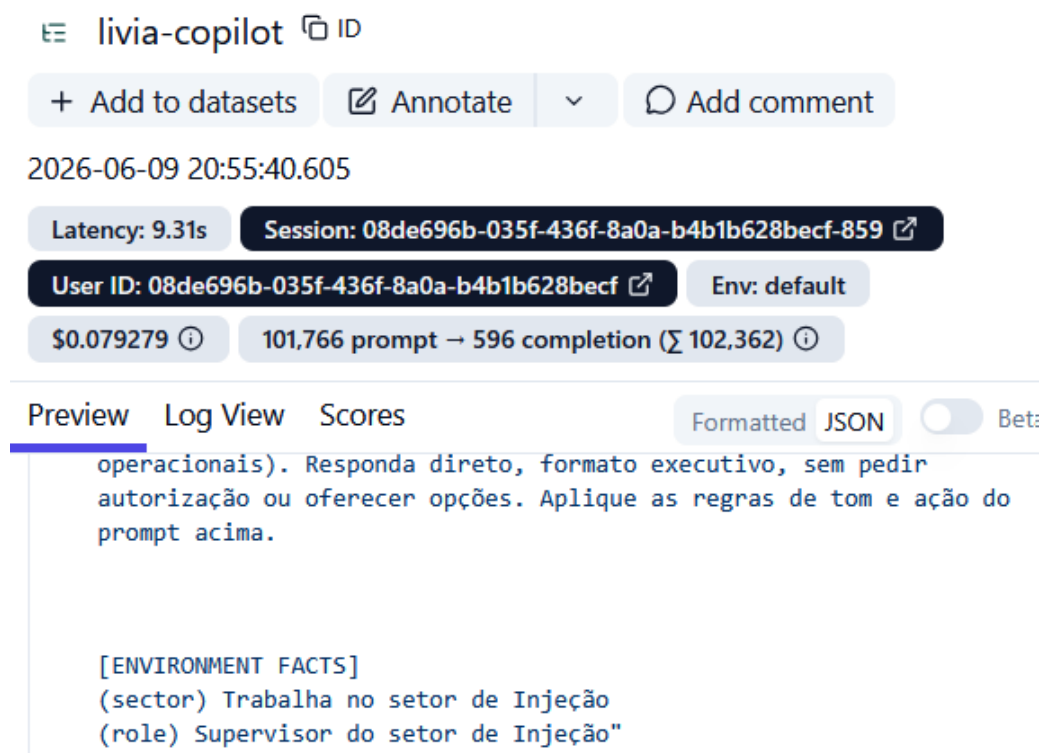
Figura 28 – Nova sessão com cache limpo: agente responde com o OEE da Injetora sem que o operador informe o setor.



Fonte: Elaborado pelo autor.

A Figura 29 confirma que o bloco [ENVIRONMENT FACTS] foi injetado corretamente no *system prompt* com os fatos persistidos da sessão anterior.

Figura 29 – *System prompt* da nova sessão com o bloco [ENVIRONMENT FACTS] injetado a partir dos fatos persistidos.

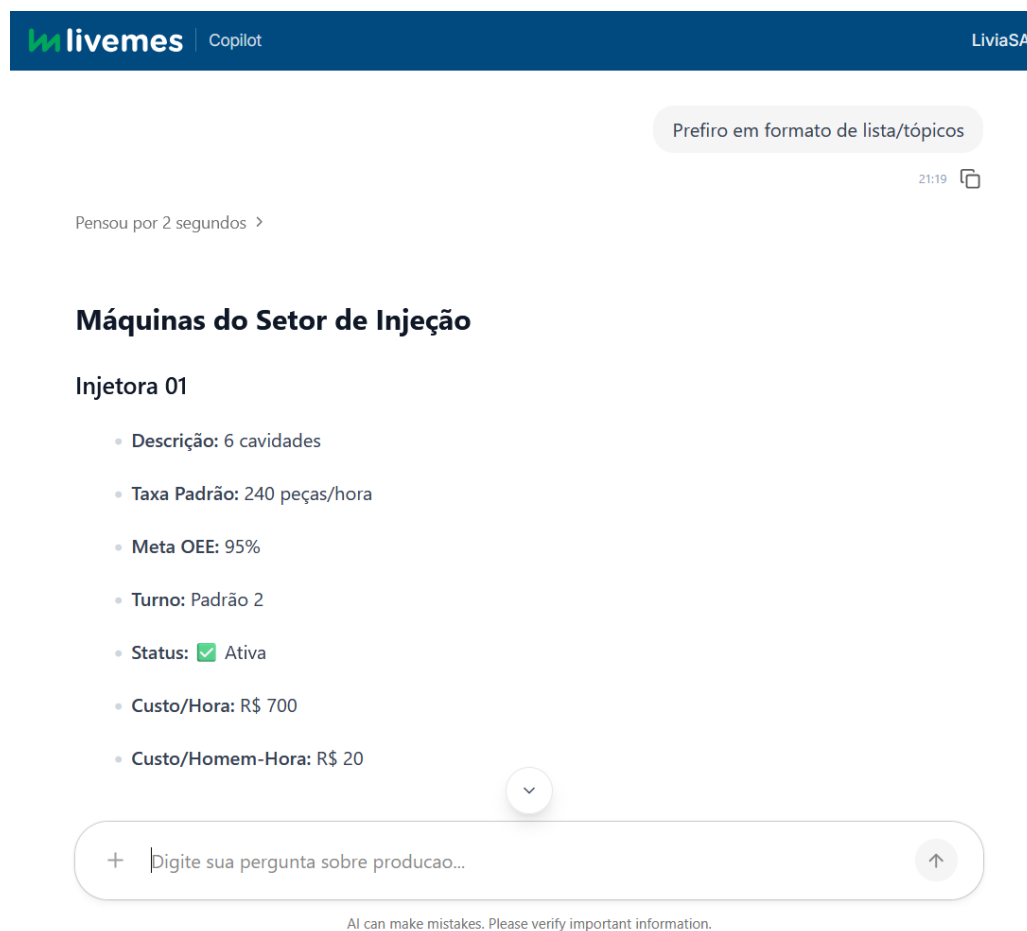


Fonte: Elaborado pelo autor.

#### 5.4.2.3 Extração e Injeção de Preferências do Usuário

O terceiro exemplo mostra o mesmo ciclo de extração aplicado às preferências do usuário. O fluxo de detecção, extração e persistência é idêntico ao descrito para os fatos do ambiente, o que muda é apenas o tipo de memória capturada, por isso não serão mostrados todos os passos detalhadamente. Foi realizada uma pergunta sobre as máquinas do setor de Injeção, e depois da resposta do agente foi enviada uma outra mensagem contendo uma declaração de preferência de formato, como mostra a Figura 30.

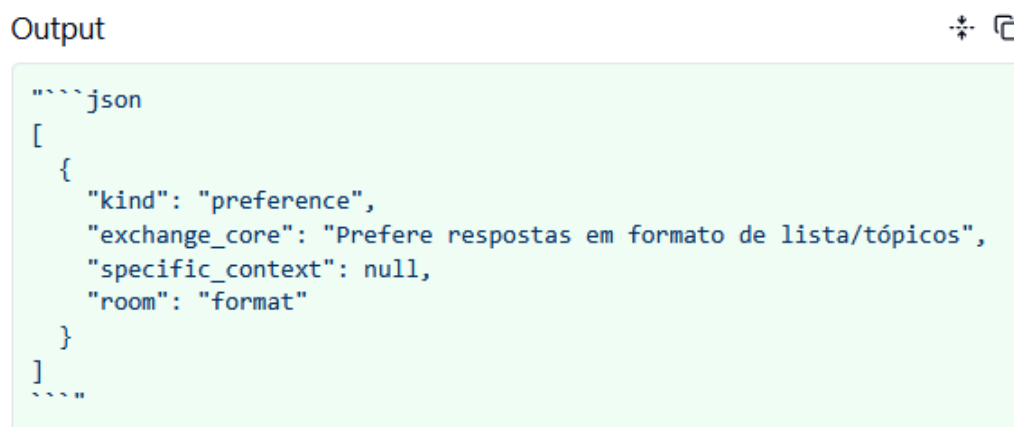
Figura 30 – Operador declara preferência de formato de resposta ao agente.



Fonte: Elaborado pelo autor.

O agente estrutura de acordo com a preferência e depois é acionado o extrator pelo regex. A Figura 31 mostra a saída do extrator, com a resposta estruturada no formato em que será salva no banco. No campo *kind* está categorizada o tipo da memória.

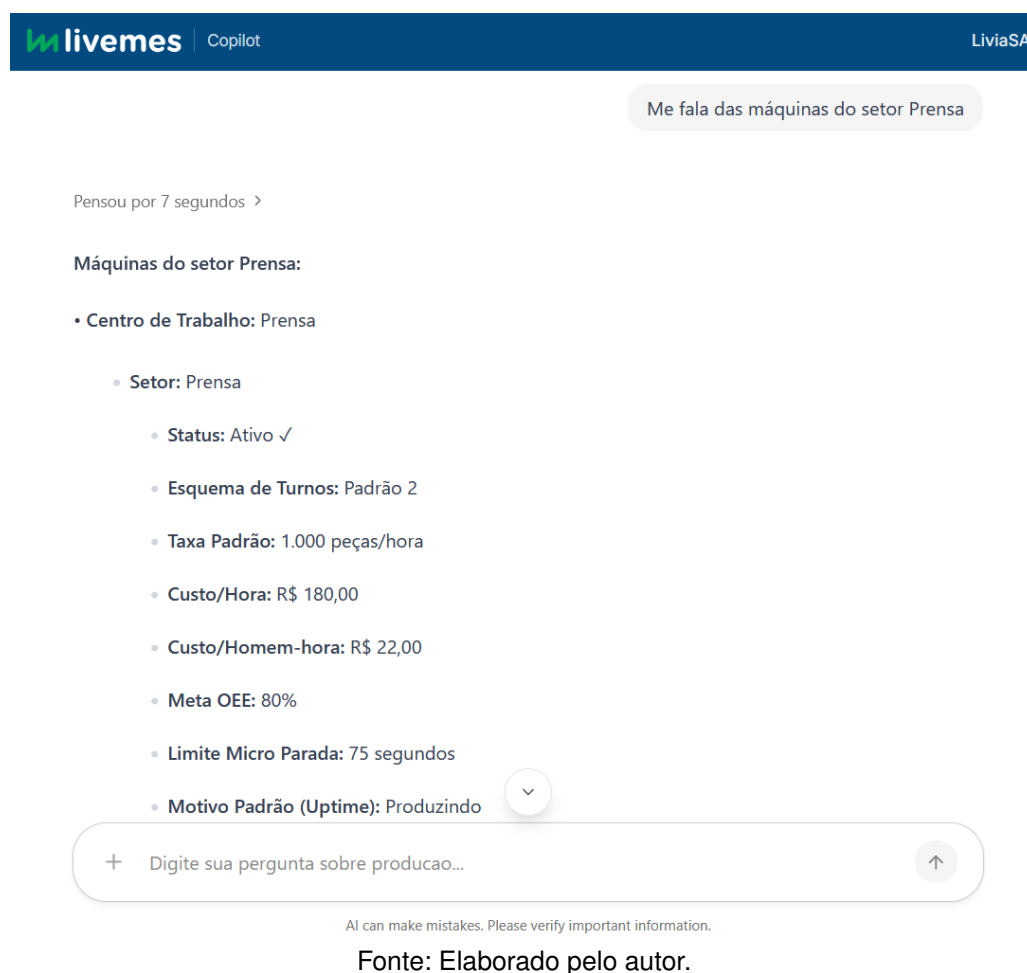
Figura 31 – Saída do extrator com a preferência de formato identificada e estruturada.



Fonte: Elaborado pelo autor.

Em uma nova sessão com cache limpo, o operador faz um pedido sem repetir a preferência. O agente responde já seguindo o formato preferido, como mostra a Figura 32, confirmando que a preferência foi recuperada do banco e injetada no contexto.

Figura 32 – Nova sessão: agente aplica a preferência de formato sem que o operador precise repeti-la.



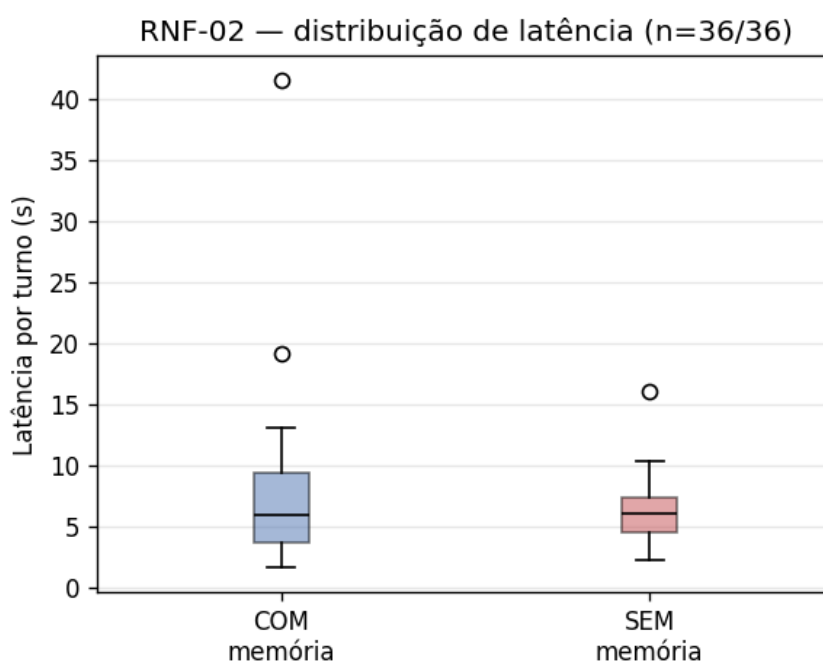
### 5.4.3 Avaliação de Eficiência e Custo

A avaliação compara o sistema em duas configurações diferentes mas rodando os mesmos cenários com 12 turnos e 3 repetições (36 amostras por condição). O número de turnos foi limitado para conter o custo operacional do experimento, dado que cada turno envolve chamadas reais ao modelo. A condição SEM é a configuração de referência sem memória, com a janela deslizante ajustada para um valor alto o suficiente para não acionar a compactação, garantindo que o histórico completo seja reenviado a cada turno, e com a memória de longo prazo desligada. A configuração COM ativa o sistema completo, com janela deslizante de 6 mensagens, para que a compactação seja acionada em poucos turnos, e memória de longo prazo habilitada. As métricas foram coletadas via Langfuse, removendo da contagem as chamadas

internas do LLM sumarizador e do extrator de memória para que o custo dessas operações não contaminem a contagem por turno, que deve refletir apenas a chamada principal de resposta ao operador.

**Latência (RNF-02).** O p50 é a mediana e o p95 é o percentil 95, valor abaixo do qual 95% das amostras se situam, refletindo o comportamento nos casos mais lentos. A mediana do COM (6,05 s) ficou próxima à do SEM (6,16 s), como ilustra a Figura 33. O p95 do COM (14,62 s) superou o do SEM (9,78 s), o que indica alguma degradação nos casos mais lentos, esperada dado que a injeção de contexto de memória aumenta o volume de tokens processados pelo modelo. Com 36 amostras, o p95 é sensível a valores extremos, tornando o p50 a métrica mais robusta para esse tamanho de amostra. Ainda assim, a degradação observada é pequena em termos absolutos, e os benefícios de continuidade e personalização que a memória ativa oferece superam o custo de latência adicional.

Figura 33 – Distribuição de latência por turno (COM vs SEM,  $n = 36$  por condição).

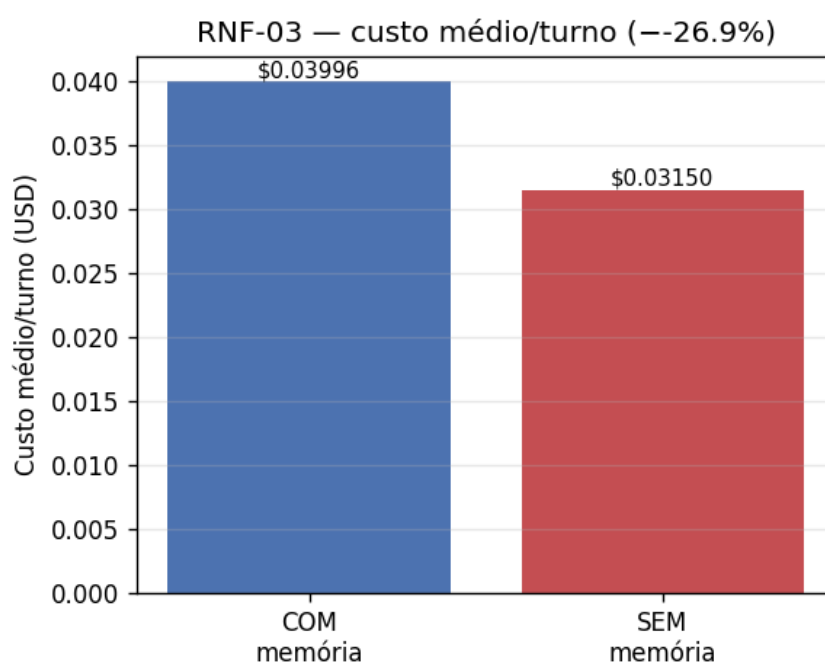


Fonte: Elaborado pelo autor.

**Custo por turno (RNF-03).** O custo médio por turno do COM (\$0,0400) ficou 26,9% acima do SEM (\$0,0315), como mostra a Figura 34. Em valor absoluto, a diferença é de apenas \$0,01 por turno. Mesmo assim, o resultado pode refletir a variabilidade natural das respostas do LLM: com apenas 3 repetições por cenário, pequenas diferenças no comprimento das respostas, no número de chamadas a ferramentas ou na forma como o modelo estrutura a saída em cada rodada podem acumular e distorcer a comparação. Esse tipo de variação é esperado em amostras pequenas e não indica que a configuração COM seja sistematicamente mais cara. Em sessões curtas, o custo

da sumarização e da extração ainda não é compensado pela redução de histórico que a janela deslizante proporciona. Parte da razão é que provedores como a Anthropic armazenam em cache os tokens que se repetem entre chamadas consecutivas. O *system prompt*, que representa o maior bloco de tokens de entrada ( $\sim 90$  k tokens neste experimento), é quase integralmente servido como leitura de cache entre os turnos, resultando em custo médio ponderado medido de \$0,245/Mtok, com cerca de 92% dos tokens servidos como leitura de cache. O histórico de conversa, por sua vez, muda a cada turno com a janela deslizante e não se beneficia desse mecanismo na mesma proporção. Isolar o *system prompt* para medir seu custo separado não é viável, pois ele define o comportamento e as ferramentas do agente e não pode ser removido sem comprometer o funcionamento. A redução de custo por turno não se confirma neste experimento de curto prazo, mas a projeção da Tabela 1 indica que o cruzamento entre as configurações ocorre em torno de 238 turnos, a partir do qual o COM passa a ser mais barato.

Figura 34 – Custo médio por turno (COM vs SEM), por rodada.



Fonte: Elaborado pelo autor.

**Tokens de entrada e eficiência de contexto (RNF-01).** A contagem total de tokens de entrada por turno foi dominada pelo prefixo estático do *system prompt* ( $\sim 90$  k tokens, servido como leitura de cache) e pelos resultados retornados das ferramentas, tornando o sinal do histórico de conversa ( $\sim 306$  tokens/turno) menor que 1% do total. Com 12 turnos, o experimento não é suficiente para comprovar empiricamente a eficiência de contexto. O que se pode afirmar é que as duas condições diferem exclusivamente na componente de histórico: em ambas, as mensagens anteriores

são reenviadas a cada turno, mas no SEM o histórico cresce sem limite enquanto no COM a janela deslizante estabiliza o histórico após atingir o limite de  $N$  mensagens, sendo  $N = 6$  neste experimento. Esse comportamento segue diretamente do design do sistema, não de uma medição direta. A Tabela 1 ilustra a divergência esperada com base nos  $\sim 306$  tokens/turno medidos: enquanto o SEM cresce linearmente, o COM permanece próximo de 917 tokens/turno.

Tabela 1 – Projeção da componente de histórico de conversa por turno (estimativa por extrapolação).

Turnos	SEM (tokens/turno)	COM (tokens/turno)	Redução
12	3.667	917	75%
25	7.640	917	88%
50	15.279	917	94%
100	30.558	917	97%

Fonte: Elaborado pelo autor, com tokenizador *cl100k\_base*.

Em 100 turnos projetados, o histórico do SEM chegaria a  $\sim 30$  k tokens/turno; somado ao prefixo estático de  $\sim 90$  k, a sessão se aproximaria cada vez mais do limite de contexto do modelo e entraria na região de degradação discutida no Capítulo 3. O COM, por design, permanece limitado independentemente da duração da sessão.

Os resultados apresentados neste capítulo, a verificação determinística dos requisitos funcionais, os exemplos de operação e a avaliação de eficiência e custo, fundamentam as conclusões discutidas no Capítulo 6, que sintetiza os objetivos atingidos, as limitações identificadas e os trabalhos futuros.

## 6 CONCLUSÃO

Este capítulo apresenta as conclusões do trabalho, retomando os objetivos definidos, discutindo as principais limitações da solução desenvolvida, os impactos para o produto e para a empresa e as direções de extensão para trabalhos futuros.

A Seção 6.1 retoma os objetivos do projeto e avalia em que medida foram atingidos. A Seção 6.2 discute as principais limitações enfrentadas durante o desenvolvimento. A Seção 6.3 apresenta os impactos do trabalho. Por fim, a Seção 6.4 aponta possibilidades de extensão para trabalhos futuros.

### 6.1 OBJETIVOS ATINGIDOS

O objetivo geral deste trabalho era projetar, desenvolver e avaliar tecnicamente um sistema de memória para um agente de IA, contemplando o gerenciamento de contexto de curto prazo e a persistência de memórias de longo prazo. Os quatro objetivos específicos definidos foram atingidos.

O primeiro objetivo, revisar e mapear a literatura sobre sistemas de memória para agentes de IA, foi atendido no Capítulo 2. A revisão cobriu os conceitos de janela de contexto, as técnicas de compactação de histórico, as taxonomias de memória de agente e os mecanismos de recuperação híbrida, estabelecendo a base teórica para as decisões de projeto do Capítulo 4.

O segundo objetivo, desenvolver o módulo de Memória de Sessão, foi atendido com a implementação do pipeline de compactação acionado por dois gatilhos independentes (limite de mensagens no buffer ou limite de tokens), do sumário estruturado em XML com campos persistidos por sessão e do mecanismo de continuidade por tópico. O módulo mantém o buffer dentro do limite configurado e opera de forma assíncrona, sem adicionar latência ao turno de resposta.

O terceiro objetivo, desenvolver o módulo de Memória de Longo Prazo, foi atendido com a implementação do extrator de preferências e fatos do ambiente, do controle de admissão de novas memórias, dos mecanismos de deduplicação e atualização e da recuperação híbrida de memórias. O módulo persiste memórias fixas por usuário e empresa e as injeta no contexto de cada nova sessão.

O quarto objetivo, realizar uma avaliação de desempenho, foi atendido com a calibração empírica dos limiares  $T_{high}$  e  $T_{low}$  da política de admissão, com a verificação dos requisitos funcionais por testes determinísticos em dois níveis integrados ao repositório, com os exemplos de operação da Seção 5.4.2 que demonstram o ciclo completo de extração, persistência e recuperação em situações reais, e com a avaliação de latência e custo por turno descrita na Seção 5.4.3, que comparou as configurações COM e SEM em 36 amostras.

## 6.2 LIMITAÇÕES ENCONTRADAS

A memória de longo prazo foi delimitada à extração de preferências do usuário e fatos do ambiente operacional. A memória experiencial, que capturaria lições aprendidas ao decorrer de sessões e percebendo padrões de comportamento, foi fundamentada teoricamente na Seção 2.4.3 mas ficou fora do escopo desta entrega. Sua implementação exigiria um terceiro pipeline de extração, esquema de tabela adicional e critérios de avaliação próprios, o que tornaria o escopo indefensável dentro do prazo disponível.

O dataset de calibração dos limiares de admissão é sintético e de pequeno porte: são apenas 16 triplas rotuladas em português, construídas por uma supervisão fraca. Os limiares  $T_{\text{high}} = 0,90$  e  $T_{\text{low}} = 0,59$  foram obtidos por esse método e são dependentes do modelo de *embedding* utilizado. Uma calibração com dados reais extraídos de *user\_preferences* e *environment\_facts* em produção poderia refinar os valores e aumentar a generalização.

A avaliação de comportamento sob carga e em sessões longas não foi conduzida neste escopo. O sistema foi validado ao longo do desenvolvimento em sessões representativas do uso típico atual, com até aproximadamente 20 turnos por sessão, de forma não estruturada como parte do processo de construção e refinamento, sem registro formal de métricas. Rodar cenários com 200 ou mais turnos teria um custo operacional elevado, o que exigiria planejamento e não é contemplado dentro do cronograma deste projeto. Por isso, o comportamento com sessões longas e bases com milhares de memórias, em especial o desempenho da busca vetorial e o custo de injeção no *prompt*, não foi medido diretamente. A medição direta da taxa de alucinação, correspondente ao RNF-04, também não foi realizada. O projeto controla a qualidade do conteúdo injetado, ou seja, o fato injetado é idêntico ao armazenado e a política de admissão semântica (Seção 5.3.3) impede fatos obsoletos ou duplicados, mas o efeito do volume adicional de tokens no contexto sobre a taxa de alucinação não foi verificado empiricamente. A medição direta exigiria um árbitro LLM com rubrica avaliadora, o que introduziria variabilidade na própria medição e está além do método determinístico adotado.

## 6.3 IMPACTOS

O sistema desenvolvido foi integrado ao produto LivIA em produção na HarboR. O principal impacto direto para o operador é a eliminação da necessidade de repetir informações a cada nova sessão. Preferências de formato, setor de atuação, função e fatos operacionais estabelecidos em interações anteriores são recuperados automaticamente e injetados no contexto, tornando a conversa mais contínua e natural.

Para o produto, o mecanismo de compactação é projetado para conter o cresci-

mento do histórico de conversa em sessões longas, o que tende a manter o contexto dentro do limite do modelo e preservar a qualidade das respostas, embora esse comportamento não tenha sido medido diretamente em sessões longas neste trabalho. A política de admissão semântica evita a acumulação de memórias redundantes ou contraditórias, mantendo o contexto injetado coerente com o estado mais recente do operador.

## 6.4 TRABALHOS FUTUROS

A primeira extensão natural é a implementação da memória experiencial. Como descrito na Seção 2.4.3, esse subsistema capturaria lições aprendidas e padrões de comportamento ao decorrer das sessões. Sua implementação constava no planejamento inicial do projeto, mas não foi possível dentro do prazo disponível.

A injeção de preferências e fatos do ambiente hoje é integral, com todos os registros do par usuário/empresa inseridos no *prompt*, limitados apenas por um corte fixo controlado pelos parâmetros `max_preferences` e `max_facts`. Esse limite é cego ao contexto da conversa, memórias pouco relevantes para a mensagem atual ocupam o mesmo espaço que as mais pertinentes, independentemente do que está sendo discutido. Com bases maiores, o custo cresce proporcionalmente ao número de registros, não à relevância deles. A substituição por uma recuperação semântica que selecione apenas as memórias mais próximas da mensagem atual tornaria a injeção sensível ao contexto e conteria o crescimento do custo mesmo com bases de memória grandes.

O sistema não possui política de esquecimento. Os resumos de sessão acumulam indefinidamente no banco, embora o impacto seja menor por não serem injetados automaticamente no contexto, apenas acessados sob demanda. Já as preferências e os fatos do ambiente persistem indefinidamente até serem contraditados, o que faz um fato sazonal como “este mês estou no turno da noite” virar verdade permanente. Duas direções complementares formam a estratégia de esquecimento: uma ferramenta *forget* que o agente possa chamar para deletar uma memória específica, e o esquecimento por decaimento, com retenção baseada em importância e em recência, o tempo decorrido desde o último acesso à memória, alinhada à abordagem de Zhong *et al.* (2024).

A calibração dos limiares de admissão foi realizada com dataset sintético. O trabalho futuro é recalibrar com dados reais de produção quando houver volume suficiente em *user\_preferences* e *environment\_facts*, substituindo as triplas sintéticas por pares extraídos da operação real. Uma extensão possível é a calibração multilíngue para operações em inglês e espanhol.

Por fim, a avaliação em produção é necessária para validar o comportamento do sistema em sessões com muitos turnos, bases com milhares de memórias e múltiplos usuários simultâneos. Esse estudo permitiria verificar se a política de admissão ainda

recupera o vizinho correto, se a latência de injeção se mantém dentro do requisito não funcional estabelecido e qual o custo real dessas sessões longas.

## REFERÊNCIAS

- ANTHROPIC. **Context windows**. 2026. Disponível em: <https://platform.claude.com/docs/en/build-with-claude/context-windows>. Acesso em: 9 mai. 2026.
- BALARABE, Tahir. **What is LLM Tokenization? A Guide to Language Model Efficiency**. 2025. Disponível em: <https://medium.com/@tahirbalarabe2/what-is-llm-tokenization-a-guide-to-language-model-efficiency-1b4ae57c180b>. Acesso em: 9 mai. 2026.
- BROWN, Tom B. *et al.* Language Models are Few-Shot Learners. **Advances in Neural Information Processing Systems**, 2020. Disponível em: <https://arxiv.org/abs/2005.14165>. Acesso em: 9 mai. 2026.
- DU, Pengfei. Memory for Autonomous LLM Agents: Mechanisms, Evaluation, and Emerging Frontiers. **arXiv preprint arXiv:2603.07670**, 2026. arXiv: 2603.07670. Disponível em: <https://arxiv.org/abs/2603.07670>. Acesso em: 5 mai. 2026.
- DU, Yufeng *et al.* Context Length Alone Hurts LLM Performance Despite Perfect Retrieval. **arXiv preprint arXiv:2510.05381**, 2025. arXiv: 2510.05381. Disponível em: <https://arxiv.org/abs/2510.05381>. Acesso em: 5 mai. 2026.
- HARBOR INTELIGÊNCIA INDUSTRIAL. **LiveMES**. 2026a. Disponível em: <https://www.harbor.com.br/livemes/>. Acesso em: 5 mai. 2026.
- HARBOR INTELIGÊNCIA INDUSTRIAL. **Sobre nós**. 2026b. Disponível em: <https://www.harbor.com.br/sobre-nos/>. Acesso em: 5 mai. 2026.
- HONG, Kelly; TROYNIKOV, Anton; HUBER, Jeff. **Context Rot: How Increasing Input Tokens Impacts LLM Performance**. 2025. Disponível em: <https://research.trychroma.com/context-rot>. Acesso em: 26 mai. 2026.
- HU, Yuyang *et al.* Memory in the Age of AI Agents: A Survey. **arXiv preprint arXiv:2512.13564**, 2026. arXiv: 2512.13564. Disponível em: <https://arxiv.org/abs/2512.13564>. Acesso em: 9 mai. 2026.

IBM. **O que é inteligência artificial (IA)?** 2026a. Disponível em:

<https://www.ibm.com/br-pt/think/topics/artificial-intelligence>. Acesso em: 9 mai. 2026.

IBM. **O que são modelos de linguagem grandes (LLMs)?** 2026b. Disponível em:

<https://www.ibm.com/br-pt/think/topics/large-language-models>. Acesso em: 9 mai. 2026.

LEWIS, Sydney. Structured Distillation for Personalized Agent Memory: 11x Token Reduction with Retrieval Preservation. **arXiv preprint arXiv:2603.13017**, 2026. arXiv: 2603.13017. Disponível em: <https://arxiv.org/abs/2603.13017>. Acesso em: 22 mai. 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. **arXiv preprint arXiv:2307.03172**, 2023. arXiv: 2307.03172. Disponível em: <https://arxiv.org/abs/2307.03172>. Acesso em: 5 mai. 2026.

NOUS RESEARCH. **Hermes Agent**. 2026. Disponível em:

<https://github.com/nousresearch/hermes-agent>. Acesso em: 26 mai. 2026.

OPENAI. **Tokenizer**. 2026. Disponível em:

<https://platform.openai.com/tokenizer>. Acesso em: 15 jun. 2026.

PACKER, Charles; FANG, Vivian; PATIL, Shishir G.; LIN, Kevin; WOODERS, Sarah; GONZALEZ, Joseph E. MemGPT: Towards LLMs as Operating Systems. **arXiv preprint arXiv:2310.08560**, 2023. arXiv: 2310.08560. Disponível em:

<https://arxiv.org/abs/2310.08560>. Acesso em: 9 mai. 2026.

PARK, Joon Sung; O'BRIEN, Joseph C.; CAI, Carrie J.; MORRIS, Meredith Ringel; LIANG, Percy; BERNSTEIN, Michael S. Generative Agents: Interactive Simulacra of Human Behavior. *In: PROCEEDINGS of the 36th Annual ACM Symposium on User Interface Software and Technology*. [S.l.: s.n.], 2023. DOI: 10.1145/3586183.3606763. Disponível em: <https://arxiv.org/abs/2304.03442>. Acesso em: 9 mai. 2026.

REDAÇÃO STARTSE. **Por que você deveria entender sobre RAG (Retrieval-Augmented Generation)?** 2024. Disponível em:

<https://www.startse.com/artigos/por-que-voce-deveria-entender-sobre-rag-retrieval-augmented-generation/>. Acesso em: 22 jun. 2026.

RUSSELL, Stuart J.; NORVIG, Peter. **Artificial Intelligence: A Modern Approach**. 3. ed. Upper Saddle River, NJ: Prentice Hall, 2010.

SENNRICH, Rico; HADDOW, Barry; BIRCH, Alexandra. Neural Machine Translation of Rare Words with Subword Units. *In: PROCEEDINGS of the 54th Annual Meeting of the Association for Computational Linguistics*. [S.l.: s.n.], 2016. P. 1715–1725.

Disponível em: <https://aclanthology.org/P16-1162/>. Acesso em: 9 mai. 2026.

TAN, Zhen *et al.* In Prospect and Retrospect: Reflective Memory Management for Long-term Personalized Dialogue Agents. **arXiv preprint arXiv:2503.08026**, 2025. arXiv: 2503.08026. Disponível em: <https://arxiv.org/abs/2503.08026>. Acesso em: 22 mai. 2026.

TAO, Dehao; MA, Guoliang; HUANG, Yongfeng; JIANG, Minghu. Membox: Weaving Topic Continuity into Long-Range Memory for LLM Agents. **arXiv preprint arXiv:2601.03785**, 2026. arXiv: 2601.03785. Disponível em: <https://arxiv.org/abs/2601.03785>. Acesso em: 22 mai. 2026.

AG-UI-PROTOCOL. **AG-UI: the Agent-User Interaction Protocol**. 2025. Disponível em: <https://github.com/ag-ui-protocol/ag-ui>. Acesso em: 7 jun. 2026.

VASWANI, Ashish; SHAZEER, Noam; PARMAR, Niki; USZKOREIT, Jakob; JONES, Llion; GOMEZ, Aidan N.; KAISER, Lukasz; POLOSUKHIN, Illia. Attention Is All You Need. **Advances in Neural Information Processing Systems**, 2017. Disponível em: <https://arxiv.org/abs/1706.03762>. Acesso em: 9 mai. 2026.

WANG, Lei *et al.* A Survey on Large Language Model Based Autonomous Agents. **Frontiers of Computer Science**, v. 18, n. 6, p. 186345, 2024. DOI: 10.1007/s11704-024-40231-1. Disponível em: <https://link.springer.com/article/10.1007/s11704-024-40231-1>. Acesso em: 9 mai. 2026.

YAO, Shunyu; ZHAO, Jeffrey; YU, Dian; DU, Nan; SHAFRAN, Izhak; NARASIMHAN, Karthik; CAO, Yuan. ReAct: Synergizing Reasoning and Acting in Language Models. **arXiv preprint arXiv:2210.03629**, 2022. arXiv: 2210.03629. Disponível em: <https://arxiv.org/abs/2210.03629>. Acesso em: 9 mai. 2026.

- ZHANG, Guilin; JIANG, Wei; WANG, Xiejiashan; BEHR, Aisha; ZHAO, Kai; FRIEDMAN, Jeffrey; CHU, Xu; ANOUN, Amine. Adaptive Memory Admission Control for LLM Agents. **arXiv preprint arXiv:2603.04549**, 2026. arXiv: 2603.04549. Disponível em: <https://arxiv.org/abs/2603.04549>. Acesso em: 4 jun. 2026.
- ZHANG, Zeyu; BO, Xiaohe; MA, Chen; LI, Rui; CHEN, Xu; DAI, Quanyu; ZHU, Jieming; DONG, Zhenhua; WEN, Ji-Rong. A Survey on the Memory Mechanism of Large Language Model based Agents. **arXiv preprint arXiv:2404.13501**, 2024. arXiv: 2404.13501. Disponível em: <https://arxiv.org/abs/2404.13501>. Acesso em: 21 mai. 2026.
- ZHAO, Wayne Xin *et al.* A Survey of Large Language Models. **arXiv preprint arXiv:2303.18223**, 2023. arXiv: 2303.18223. Disponível em: <https://arxiv.org/abs/2303.18223>. Acesso em: 9 mai. 2026.
- ZHONG, Wanjun; GUO, Lianghong; GAO, Qiqi; WANG, He; WANG, Yanlin. MemoryBank: Enhancing Large Language Models with Long-Term Memory. **Proceedings of the AAAI Conference on Artificial Intelligence**, v. 38, n. 17, p. 19724–19732, 2024. DOI: 10.1609/aaai.v38i17.29946. Disponível em: <https://arxiv.org/abs/2305.10250>. Acesso em: 9 mai. 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. **arXiv preprint arXiv:2506.15841**, 2025. arXiv: 2506.15841. Disponível em: <https://arxiv.org/abs/2506.15841>. Acesso em: 22 mai. 2026.