Implementação de IA Generativa com LLM

Metodologia, implementação e acompanhamento de projetos de IA

Junho 2026

MINISTÉRIO DA GESTÃO E DA INOVAÇÃO EM SERVIÇOS PÚBLICOS

SUMÁRIO

SUMÁRIO

1. Introdução

Este documento é parte integrante do **GuIA** (**G**uia **u**nificado de **I**nteligência **A**rtificial). Este não é um documento descritivo de caso de uso de tecnologia, mas sim um anexo técnico, que se destina a orientar a área técnica com boas práticas, recomendações e requisitos obrigatórios para a implementação da solução tecnológica identificada como adequada a um desafio.

Leitura orientada por perfil

Arquitetos e desenvolvedores devem priorizar fundamentos, arquitetura, técnicas, *guardrails* e modos de falha; gestores de produto e áreas demandantes devem priorizar escopo, aplicabilidade, métricas e critérios de aceite; segurança, privacidade e conformidade devem priorizar riscos, tratamento de dados, controles, *red teaming* e requisitos mínimos de produção. Essa trilha evita leitura linear obrigatória por todos os públicos e direciona cada perfil às seções indispensáveis.

Embora boas práticas de implementação sejam relevantes a todos os sistemas de IA, sistemas baseados em grandes modelos de linguagem possuem especificidades técnicas, como seleção e governança do modelo, engenharia e versionamento de *prompts*, controle de janela de contexto, parâmetros de decodificação, *guardrails*, avaliação de alucinações, segurança de *prompts* e integração com ferramentas. Essas especificidades demandam orientações próprias, que constituem o objeto deste anexo.

Este anexo referencia atividades específicas em várias etapas do documento *core* (**GuIA**) no ciclo de vida de uma solução de IA, desde sua ideação até sua descontinuidade, representada na Figura 1.

Figura 1 - Ciclo de vida de soluções de IA

Regras de Aplicação e Integração Modular

Este anexo complementa o documento-base (GuIA) e o *template* de Caso de Uso sempre que o projeto envolver IA Generativa textual (LLM), com ou sem engenharia de *prompt*. Integra o **GuIA** como Anexo Técnico, complementando as diretrizes gerais do documento principal e servindo de referência obrigatória para casos de uso que envolvam o uso de modelos de linguagem de grande escala.

Para manter a clareza das responsabilidades técnicas, a fusão dos módulos não é recomendada. A integração ocorre de forma coordenada por meio de referências cruzadas e critérios comuns de risco, divididos em três frentes:

- **Este Anexo (LLM):** Regula o comportamento, operação, avaliação e governança do modelo.

- **Anexo de Base de Conhecimento para soluções de IA:** Governa a camada de conteúdo, curadoria, metadados, proveniência e controle de acesso aos dados.
- **Anexo de RAG (Geração aumentada por recuperação):** Trata da recuperação de documentos externos e orquestração do contexto quando o sistema precisar consultar bases externas.

2. Escopo e aplicabilidade

Este anexo insere-se no contexto do **Plano Brasileiro de Inteligência Artificial 2024-2028 (PBIA)**, que estabelece diretrizes estratégicas para o desenvolvimento e aplicação responsável de IA no país, com ênfase em transparência, segurança e conformidade regulatória no setor público. Este anexo possui alinhamento com as orientações do Tribunal de Contas da União (TCU) sobre uso e auditoria de sistemas de IA generativa e adota como referências complementares padrões internacionais reconhecidos como a **ISO/IEC 42001:2023** (Sistemas de Gestão de IA), **ISO/IEC 42005:2025** (Avaliação de Impacto de Sistemas de IA) e o ***NIST AI Risk Management Framework***. Estes padrões fornecem um *framework* de conformidade estruturado para avaliação, implementação e operação de sistemas de IA no setor público. O documento adota como base normativa obrigatória o Decreto 9.637/2018 (Política Nacional de Segurança da Informação), a IN SGD/ME nº 1/2019 (requisitos de segurança para contratações de soluções de TIC) e a IN GSI/PR nº 1/2020 (gestão de segurança da informação na administração pública federal), bem como a ISO/IEC 27001:2022 (gestão de segurança da informação) como norma base do framework de segurança, complementar às ISO/IEC 27701:2019 e ISO/IEC 42001:2023 já referenciadas.

A crescente adoção de sistemas de IA Generativa no setor público brasileiro exige diretrizes técnicas específicas para garantir conformidade regulatória e segurança das operações. Modelos de linguagem de grande escala (LLMs – Large Language Models) apresentam riscos conhecidos de gerar informações imprecisas ou inventadas (alucinações), particularmente em domínios que exigem precisão factual e rastreabilidade de fontes.

Neste contexto, o presente anexo aplica-se a sistemas baseados em modelos de IA Generativa de texto (LLMs) operando sem acoplamento com bases externas de busca (IA Generativa "pura"). Esta arquitetura permite:

- Geração de texto em linguagem natural (criativa, técnica ou institucional);
- Sumarização, tradução e classificação *zero-shot/few-shot*;
- Inferência sequencial, extração de informação e resposta a perguntas fundamentadas no conteúdo fornecido no próprio *prompt*;
- Ajuste fino supervisionado e por reforço para domínios específicos (*fine-tuning*).

Porém, há limitações de escopo. Este anexo **não se aplica** a:

- Sistemas que requerem rastreabilidade explícita de fontes documentais externas (cobertos pelo Anexo RAG);
- Sistemas puramente de busca ou recuperação, sem geração de texto;
- Sistemas de *Machine Learning* clássico, sem geração de linguagem natural.

3. Ciclo de vida sugerido para a solução

3.1 Impactos no ciclo de vida do sistema

Na etapa de Prospecção de Desafios, sistemas LLM exigem análise de viabilidade centrada nas tarefas cognitivas alvo, avaliando se o conhecimento paramétrico do modelo-base é suficiente para o domínio ou se será necessário *fine-tuning*. Deve ser realizado inventário das interações esperadas (tipos de consulta, perfis de usuário, volumes) e avaliação de sensibilidade dos dados trafegados. Realiza-se a Análise de Impacto à Proteção de Dados (RIPD) como parte do Estudo Técnico Preliminar, conforme previsto no Art. 5º, inciso XVII da LGPD, considerando dados pessoais que possam ser inseridos por usuários ou incluídos nos *prompts* da aplicação. A avaliação de risco ético desta etapa incorpora ameaças específicas de sistemas LLM, como injeção direta e indireta de *prompt*, geração de conteúdo nocivo ou enviesado, uso inseguro de ferramentas e exposição de dados pessoais a provedores externos, aplicando metodologia STRIDE adaptada, com foco nas ameaças *Spoofing* (impersonificação via instrução adversarial), Tampering (manipulação do *system prompt* em trânsito) e *Information Disclosure* (exfiltração do *system prompt* por usuário), conforme guia OWASP *Threat Modeling* e OWASP Top 10 for LLM *Applications*.

Na etapa de Estruturação, aprofunda-se a compreensão do desafio com foco na viabilidade e nos riscos associados ao uso do LLM puro. Define-se a estratégia de obtenção do modelo (API proprietária, *open-source* auto-hospedado ou *fine-tuning*), estimando custos de inferência por token e requisitos de soberania de dados. A decisão "*Make or Buy*" considera a disponibilidade de dados curados para eventual *fine-tuning*, a maturidade dos processos de *prompt engineering* da equipe e os requisitos de latência e disponibilidade. Quando APIs externas forem utilizadas, devem ser identificados os dados enviados ao provedor, incluindo consultas, histórico de conversa, documentos inseridos no *prompt*, dados de treinamento para *fine-tuning* e conteúdo utilizado em *embeddings*. As métricas de sucesso e o plano de experimentação são definidos nesta etapa, incluindo métricas de qualidade de geração (aderência ao *prompt*, ausência de alucinações detectáveis, coerência e pertinência), métricas operacionais (TTFT, *tokens* por segundo, latência P95/P99, taxa de erro e custo por interação) e métricas de negócio relacionadas à satisfação do usuário e à redução de escalações para atendimento humano.

Na etapa de Experimentação, são prototipados e devem ser avaliados os componentes centrais do sistema LLM:

- *system prompt* inicial, com refinamento iterativo e testes de regressão;
- seleção e comparação de modelos base candidatos em termos de qualidade, custo e latência;
- configuração de parâmetros de decodificação (temperatura, *top-p*) alinhada ao nível de precisão exigido;
- prototipação de *guardrails* de entrada e saída para detecção de conteúdo nocivo e injeções adversariais; e
- quando aplicável, prototipação de *serving* local, quantização e chamadas de ferramentas.

A validação técnica inclui avaliação de qualidade de geração com conjuntos de teste curados, avaliação humana por especialistas de domínio e *red teaming* adversarial inicial para

identificar vulnerabilidades de *jailbreak*. Devem ser aplicadas as referências metodológicas OWASP Top 10 for LLM *Applications* e NIST AI *Risk Management Framework.*

Na etapa de Implementação, a arquitetura LLM validada é construída de forma robusta, escalável e segura desde a concepção (*security by design*). Toda credencial de API deve ser gerenciada exclusivamente via cofre de credenciais (*vault* ou HSM), sendo vedada sua ocorrência em código-fonte, *notebooks* ou arquivos de configuração versionados. *Tokens* de API devem ter política de rotação periódica com prazo máximo definido. O pipeline de CI/CD deve incluir SAST (análise estática de segurança) e SCA (varredura de composição de software) integrados a cada *commit*. O *system prompt* e os *prompts* de *few-shot* devem ser versionados junto ao código da aplicação. Os testes integrados, de carga e de segurança incluem tentativas de *jailbreak*, injeção de *prompt* e avaliação de comportamento sob consultas adversariais, inclusive injeção indireta em documentos, e-mails, páginas web ou campos estruturados processados pelo modelo. Os ambientes de desenvolvimento, *staging* e produção devem ser isolados por segmentação de rede. O *Model Card* do modelo é artefato obrigatório antes da entrada em produção e deve seguir template institucional ou formato compatível com práticas como *Hugging Face Model Cards* ou *Google Model Cards*, contemplando intenção de uso, usos fora de escopo, dados de treinamento, métricas de desempenho, limitações conhecidas, vieses identificados, *knowledge cutoff*, riscos residuais, requisitos de supervisão humana e responsável pela manutenção. O sistema deve ser submetido a auditoria de viés (*fairness audit*) antes da implantação e periodicamente em produção, com metodologia documentada: grupos avaliados, hipóteses de risco, amostras estratificadas, métricas como paridade demográfica, igualdade de oportunidade e calibração, ferramentas utilizadas (por exemplo *Fairlearn* ou *AI Fairness* 360) e plano de remediação.

Na etapa de Implantação (Rollout), executa-se o plano de *rollout* da solução LLM, validando o processo de gestão de mudança (GMUD) para um *deploy* seguro. Ocorre a formalização da entrada em produção, com especial atenção à validação dos *guardrails* em ambiente produtivo e à verificação de que o *system prompt* versionado foi corretamente implantado. Realiza-se a transferência oficial da solução incluindo documentação dos *prompts*, configurações de parâmetros de decodificação e *runbooks* de operação para a equipe de sustentação responsável.

Na etapa de Sustentação, a gestão de sistemas LLM incorpora atividades contínuas de monitoramento e melhoria. O monitoramento contínuo acompanha métricas de qualidade de geração em tempo real, detecta deriva comportamental do modelo (especialmente após atualizações pelo fornecedor da API) e analisa o *feedback* de usuários para identificar anomalias. A gestão de mudanças é crítica: atualizações de modelos via API podem alterar o comportamento silenciosamente, exigindo testes de regressão de *prompts* automatizados a cada nova versão. Os ciclos de melhoria contínua por MLOps e FinOps incluem auditoria periódica de logs de acesso conforme Art. 46 da LGPD, monitoramento de custos de inferência por token, gestão de incidentes de conteúdo nocivo ou alucinações detectadas e conformidade com políticas de retenção e descarte alinhada à ISO/IEC 27701.

Todas as comunicações entre componentes do sistema LLM (aplicação, API do modelo e interface do usuário) devem utilizar TLS 1.2 como protocolo mínimo, com preferência por TLS 1.3. Modelos *fine-tuned* auto-hospedados devem ter política formal de *backup* com RTO e RPO definidos e testes periódicos de restauração documentados. Contêineres do sistema devem executar como usuário *non-root*, com sistema de arquivos em modo *read-only* onde aplicável e perfis restritivos de *syscall*.

Na etapa de Desativação, sistemas LLM exigem processos controlados de encerramento, incluindo revogação de tokens de API, exclusão segura de *datasets* de *fine-tuning* que contenham dados pessoais e decisões documentadas sobre retenção de logs de inferência. O plano de desativação segura (*secure decommissioning*) deve contemplar a migração para eventual solução sucessora, a preservação de evidências para auditoria conforme legislação aplicável (LGPD Art. 15 – término do tratamento de dados), o arquivamento de versões de *system prompts* e configurações, e a documentação de lições aprendidas para fins de rastreabilidade e governança.

Figura 2 - Ciclo de vida específico para LLM

A figura apresenta uma visão sintética do ciclo de vida específico para soluções com LLM, conectando planejamento, design, desenvolvimento, testes, *deploy* e evolução. Ela reforça que feedback e dados reais de uso alimentam melhorias contínuas, em alinhamento com as atividades de sustentação e evolução descritas na seção.

3.2 Atividades típicas por etapa

Compatibilidade com o GuIA: para cada etapa do ciclo de vida, os artefatos do projeto devem manter a mesma lógica de organização do GuIA, explicitando entregáveis, ponto de decisão, matriz RACI e recomendações, políticas e boas práticas. As atividades abaixo detalham os aspectos específicos de LLM, mas devem ser usadas em conjunto com essa estrutura comum.

Etapa 1 – Prospecção de Desafios

- Mapeamento das tarefas cognitivas alvo (geração, sumarização, classificação, extração, raciocínio).

- Avaliação de adequação do conhecimento paramétrico do modelo-base ao domínio institucional.
- Inventário de dados sensíveis que poderão transitar nos *prompts* (dados pessoais, informações classificadas).
- Inclusão de riscos específicos de LLM (vide seção "Riscos específicos do uso de LLM") na avaliação de risco ético do Estudo Técnico Preliminar.

Etapa 2 – Estruturação do desafio de IA no serviço público

- Escopo das tarefas cognitivas que o sistema deve executar (ex.: geração de rascunhos, sumarização de documentos inseridos no *prompt*, resposta a perguntas de domínio geral, tradução).
- Decisão "Make or Buy" considerando modelo via API, modelo *open-source* auto-hospedado ou *fine-tuning* sobre modelo base.
- Estimativa de custo de inferência por tipo de consulta e por volume esperado de usuários.
- Mapeamento de requisitos de soberania de dados e privacidade que impactam a escolha do modelo.
- Identificação dos dados enviados a APIs externas e avaliação de compatibilidade com LGPD, sigilo institucional e contratos do provedor.
- Definição de métricas técnicas de qualidade de geração (aderência ao *prompt*, ausência de toxicidade), métricas operacionais (TTFT, *tokens* por segundo, latência P95/P99) e de negócio para o plano de experimentação.
- Avaliação de viabilidade técnica e financeira da arquitetura LLM (modelo, camada de aplicação, *guardrails*).
- Definição da arquitetura de referência do caso de uso, com fronteiras entre aplicação, orquestração, modelo, segurança, ferramentas, memória e observabilidade.

Etapa 3 – Experimentação

- Prototipação e refinamento iterativo do *system prompt* com testes de *zero-shot*, *few-shot* e CoT.
- Seleção e comparação de modelos base candidatos (qualidade, latência, custo, suporte ao português).
- Avaliação de quantização, QLoRA e serving local quando houver requisito de auto-hospedagem ou soberania de dados.
- Avaliação de modelos de *embedding*, quando houver busca semântica, memória vetorial ou métricas baseadas em similaridade.
- Configuração e teste de parâmetros de decodificação (temperatura, *top-p*) para o perfil de precisão exigido.
- Prototipação de *guardrails* de entrada e saída (detecção de injeção direta e indireta de *prompt*, toxicidade, dados sensíveis).
- Prototipação de *function calling* ou uso de ferramentas, com validação de argumentos fora do modelo.

- Definição de *schemas* de saída estruturada e testes de parsing/validação "tipada".
- Avaliação de qualidade de geração (aderência ao *prompt*, consistência, ausência de alucinações mensuráveis) com conjuntos de teste curados.
- Avaliação humana por especialistas de domínio.
- *Red teaming* adversarial inicial: testes de *jailbreak*, injeção direta e indireta de *prompt*, uso indevido de ferramentas e extração do *system prompt.*
- Validação de conformidade com OWASP Top 10 for LLM *Applications*.

Etapa 4 – Implementação

- Arquitetura e design final do sistema LLM com *security by design.*
- Implementação do orquestrador com *templates* de *prompt* versionados, limites de contexto, política de *retry* e *fallback*.
- Versionamento do *system prompt* e *prompts* de *few-shot* integrado ao repositório de código.
- Implementação de *guardrails* de entrada (classificação de intenção, detecção de injeção direta e indireta) e saída (toxicidade, dados sensíveis, fundamentação).
- Implementação de validação e autorização para chamadas de ferramentas ou APIs externas.
- Implementação de controle de memória e estado conversacional, com segregação por usuário, expiração, retenção e auditoria.
- Validação de cadeia de suprimentos de modelo, tokenizador, adaptadores, contêineres e bibliotecas de *serving*.
- Gestão de credenciais de API via cofre (*vault*/HSM) com política de rotação.
- Preparação de pipeline de CI/CD com testes de regressão de *prompts* automatizados.
- Criação de painéis de monitoramento de qualidade de geração e custos de inferência, incluindo TTFT, *tokens* por segundo e latência P95/P99.
- Testes integrados, de carga, de segurança e fim a fim, incluindo ataques de *jailbreak* e avaliação de viés.
- Auditoria de conformidade com LGPD e OWASP Top 10 for LLM *Applications*.

Etapa 5 – Implantação (Rollout)

- Execução do plano de *rollout* com validação dos *guardrails* em ambiente produtivo.
- Validação do GMUD para *deploy* seguro.
- Verificação de que o *system prompt* versionado foi corretamente implantado e está protegido contra exfiltração.
- Transferência oficial da solução (documentação de *prompts*, *runbooks*, configurações) para equipe de sustentação.

Etapa 6 – Sustentação

- Monitoramento contínuo de métricas de qualidade de geração e toxicidade em produção.

- Testes de regressão de *prompts* automatizados a cada atualização de modelo pelo fornecedor.
- Detecção de deriva comportamental do modelo e análise de *feedback* de usuários.
- Gestão de mudanças: testes de compatibilidade para novas versões do modelo-base (API ou local).
- Auditoria periódica de logs de acesso (LGPD Art. 46).
- Gestão de incidentes de geração de conteúdo nocivo ou alucinações detectadas.
- Ciclos de melhoria contínua via MLOps e FinOps.
- Manutenção de *datasets* de avaliação contínua e testes sentinela para regressões de segurança, qualidade e custo.

Gestão de vulnerabilidades pós-*deploy*: o contratante deve manter processo formal de identificação, triagem e remediação de vulnerabilidades pós-*deploy*, com SLA por severidade CVSSv3 (crítico: 24 horas; alto: 7 dias; médio: 30 dias), canal de reporte formal, responsável designado pela triagem e procedimento de aplicação de *patches* emergenciais com janela de manutenção predefinida. O processo deve ser documentado no *runbook* de operação entregue na transferência da solução.

Etapa 7 – Desativação

- Identificação e planejamento da desativação dos componentes do sistema LLM (aplicação, integrações, monitoramento).
- Revogação de *tokens* de API e credenciais de acesso ao modelo.
- Exclusão segura de *datasets* de *fine-tuning* que contenham dados pessoais, conforme LGPD Art. 15.
- Migração para solução sucessora e desconexão técnica.
- Decisão documentada sobre retenção de logs de inferência versus descarte seguro.
- Preservação de evidências para auditoria (LGPD Art. 15).
- Arquivamento de versões de *system prompts* e configurações de modelos.
- Documentação de lições aprendidas e memória técnica do projeto.

4. Caracterização do sistema de IA

4.1 Fundamentos conceituais de LLMs

Um sistema de LLM puro é composto por três componentes técnicos fundamentais:

- **Modelo fundacional (*foundation model*):** responsável pela geração de texto natural. Recebe como entrada a consulta do usuário combinada com instruções do sistema (*system prompt*) e histórico de conversa, e produz resposta coerente, fluente e contextualmente adequada. Modelos podem ser obtidos via API de terceiro, auto-hospedados a partir de repositório público com processo formal de verificação de pesos, ou desenvolvidos internamente. No setor público, considerações sobre **soberania de dados**, custos de API e requisitos de privacidade influenciam a escolha entre modelos proprietários via API e modelos *open-source* auto-hospedados.
- **Tokenizador:** converte texto de entrada em unidades discretas (*tokens*) que o modelo processa, e reconverte os *tokens* de saída em texto legível. A granularidade do tokenizador afeta diretamente a janela de contexto disponível e o custo de inferência, pois APIs cobram por *token* consumido. Modelos distintos utilizam tokenizadores próprios; estimar o número de *tokens* por consulta é etapa obrigatória no planejamento de custos.
- **Camada de inferência e decodificação:** infraestrutura lógica responsável por executar o modelo, gerenciar contexto, cache de atenção, paralelismo, limites de *tokens*, critérios de parada e parâmetros de amostragem. Esta camada pode ser oferecida por API de terceiro ou por servidor de inferência auto-hospedado, e afeta diretamente latência, *throughput*, custo e superfície de segurança.

Instruções do sistema (*system prompt*): o *system prompt* não é componente arquitetural do modelo, mas sim um artefato operacional de alinhamento e controle em produção. Consiste em bloco de texto fornecido antes da conversa do usuário para configurar papel, restrições, tom, políticas de resposta e comportamento esperado. Deve ser versionado, testado contra regressões, protegido contra exfiltração e tratado como configuração crítica da aplicação.

4.2 Fluxo fundamental do LLM

O ciclo de inferência de um LLM percorre quatro etapas sequenciais:

Figura 3 - Visão do esquema operacional de referência

- **Entrada (*input*):** o texto recebido – composto pelo *system prompt*, histórico de conversa e consulta do usuário – é concatenado em uma sequência única chamada *prompt* completo.
- **Tokenização:** o *prompt* completo é segmentado em *tokens* pelo tokenizador do modelo. Cada *token* é mapeado a um índice numérico no vocabulário do modelo.
- **Inferência (mecanismo de atenção/*Transformers*):** os *tokens* são processados por camadas de atenção multi-cabeça que modelam dependências entre todas as posições da sequência. O resultado é uma distribuição de probabilidade sobre o vocabulário para o próximo *token*.
- **Decodificação e saída:** tokens são amostrados iterativamente da distribuição (via *greedy decoding*, *beam search* ou amostragem estocástica com temperatura) até critério de parada. Os tokens resultantes são detokenizados em texto legível e retornados ao usuário.

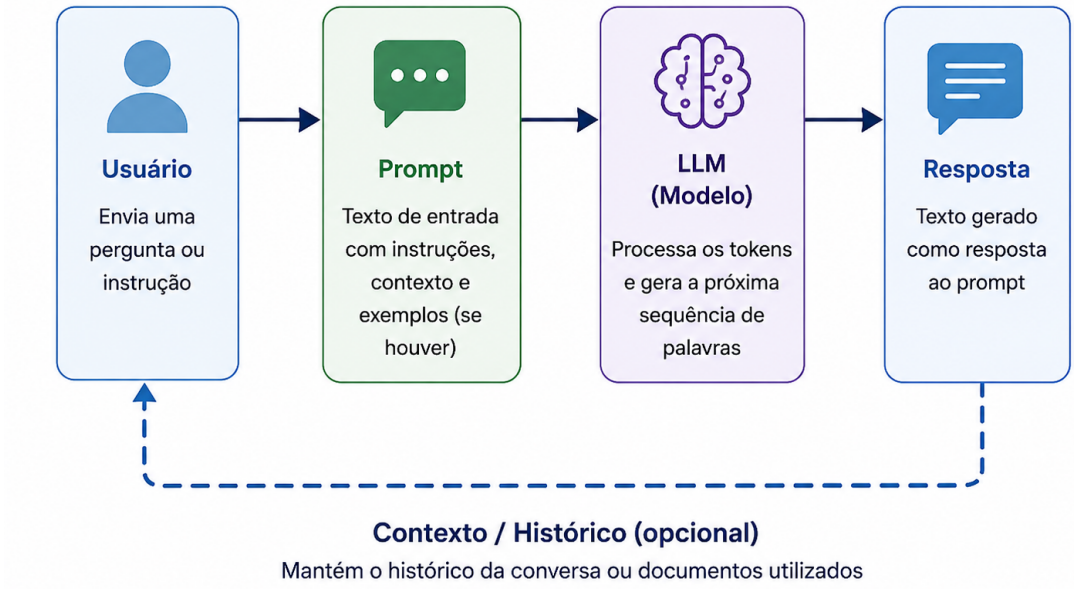

Figura 4 - Fluxo fundamental de interação com um LLM

A imagem resume o fluxo descrito na seção: o usuário fornece uma pergunta ou instrução, o *prompt* organiza instruções e contexto, o LLM processa *tokens* e gera a resposta. O retorno tracejado representa o uso opcional de histórico ou documentos no contexto, reforçando a importância de controlar o que é reaproveitado entre interações.

4.3 Aplicabilidade

Esta seção apresenta os critérios que indicam quando o **LLM puro** é a abordagem adequada. Cada critério corresponde a uma necessidade específica que esta arquitetura atende sem dependência de base de busca externa.

LLM puro é adequado quando:

- A informação necessária pode ser integralmente fornecida no *prompt* (documentos curtos, contexto da conversa);
- A tarefa é de geração criativa, tradução, sumarização ou extração estruturada sem exigência de rastreabilidade de fonte;
- O domínio é coberto pelo conhecimento paramétrico do modelo e a acurácia factual absoluta não é crítica;
- Simulação de inferência sequencial e resolução de problemas de domínio geral são o objetivo central.

A tabela a seguir orienta a decisão entre LLM puro, RAG e *fine-tuning*. A classificação é didática: em produção, arquiteturas costumam formar um contínuo entre LLM puro, enriquecimento de *prompt*, RAG, uso de ferramentas e sistemas "agentivos".

Quadro 1 – Comparativo de critérios para escolha

Critério	LLM Puro	RAG	Fine-Tuning
Rastreabilidade de fontes	Não	Sim	Não
Acesso a informação atualizada	Não	Sim	Não
Domínio especializado/institucional	Parcial	Parcial	Sim
Custo de implantação	Baixo	Médio	Alto
Latência de resposta	Baixa	Média	Baixa
Soberania de dados (modelo local)	Possível	Possível	Possível
Requer curadoria de base documental	Não	Sim	Não

Nota sobre soberania de dados: a marcação "Possível" é condicionada ao uso de componentes locais em todo o fluxo. RAG com API externa de *embedding* pode enviar consultas e trechos documentais ao provedor; *fine-tuning* em API externa pode expor o conjunto de treinamento; e LLM puro via API externa transmite ao provedor o *prompt*, histórico e dados inseridos pelo usuário. A decisão arquitetural deve registrar quais dados trafegam fora do ambiente do con-

tratante e quais garantias contratuais e técnicas mitigam esse risco.

Contínuo arquitetural: a escolha entre LLM puro, RAG e *fine-tuning* não deve ser interpretada como taxonomia rígida. Sistemas reais podem combinar:

- LLM puro, quando todo o contexto vem do usuário ou da conversa;
- *prompt augmentation*, quando a aplicação injeta regras, exemplos e contexto institucional curto;
- RAG, quando há recuperação semântica de documentos;
- sistemas ampliados por ferramentas, quando o modelo consulta APIs ou bases estruturadas; e
- agentes, quando há planejamento e execução multi-etapas. Cada avanço no contínuo aumenta a capacidade, mas também superfície de ataque, complexidade operacional, custo e necessidade de observabilidade.

Figura 5 - Geração aumentada por recuperação

A imagem ilustra a diferença entre geração baseada apenas no conhecimento paramétrico do modelo e geração apoiada por documentos recuperados. Ela complementa a discussão sobre o contínuo arquitetural, mostrando que o RAG insere uma etapa de busca semântica antes da geração da resposta.

4.4 Limitações de uso

O LLM puro apresenta limitações inerentes que devem ser consideradas na decisão arquitetural:

- **Alucinação intrínseca:** sem base de busca externa, o modelo pode fabricar fatos com aparência de confiança, especialmente quando consultado sobre assuntos específicos fora do seu conhecimento paramétrico;
- **Corte de conhecimento (*knowledge cutoff*):** o modelo não tem acesso a eventos, legisla-

ções ou dados publicados após sua data de treinamento;

- **Janela de contexto finita:** documentos que excedem o limite de tokens suportado precisam de estratégias específicas de truncamento, sumarização hierárquica ou divisão (*map-reduce*);
- **Ausência de rastreabilidade:** não é possível citar o documento original que embasou uma afirmação, o que pode ser inaceitável em contextos que exigem fundamentação normativa;
- **Custo proporcional a *tokens*:** conversas longas e documentos extensos aumentam exponencialmente o custo de inferência em APIs comerciais.

4.5 Arquitetura de referência para sistemas LLM

Uma implementação de LLM em produção deve explicitar as fronteiras entre camada de aplicação, orquestração, modelo, segurança e observabilidade. Uma arquitetura de referência mínima é:

Usuário -> API Gateway -> Orquestrador -> Validador de entrada / guardrail pre-LLM -> Prompt Builder / Template versionado -> LLM ou servidor de *inferência* -> Validador de *saída / Policy Engine* -> Ferramentas externas, quando *aplicável* -> *Formatador de resposta -> Usuário* \-> Logs, *métricas, tracing e auditoria*

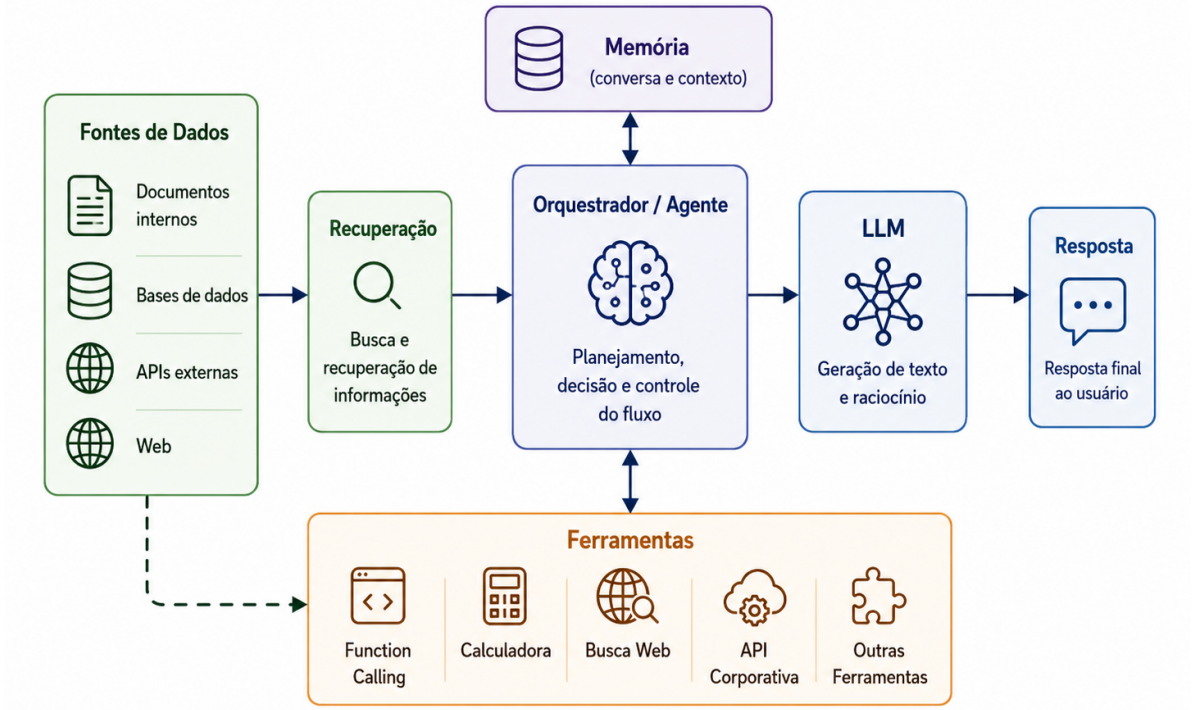

Figura 6 - Arquitetura de referência para sistemas LLM

A figura traduz o fluxo arquitetural textual em uma visão de componentes, destacando fontes de dados, recuperação, memória, orquestrador, LLM, ferramentas e resposta. Ela evidencia que o modelo é apenas uma parte da solução e que a camada de orquestração coordena decisões, integrações e controles.

Camada de aplicação: autentica o usuário, aplica autorização, controla sessão, valida dados

de entrada e define o contrato de resposta esperado. Não deve delegar ao modelo decisões de autenticação, autorização ou validação transacional.

Camada de orquestração: monta o *prompt*, seleciona modelo, aplica políticas de roteamento, chama ferramentas, controla tentativas, define *fallbacks* e registra eventos técnicos. Esta é a fronteira principal entre lógica determinística do sistema e geração probabilística do modelo.

Camada de modelo: executa inferência via API externa ou servidor local. Deve ter versão controlada, limites de *tokens*, parâmetros de decodificação documentados, isolamento de credenciais e métricas de latência, *throughput*, erro e custo.

Camada de segurança e qualidade: contém *guardrails* de entrada, contexto, saída e ação. Deve operar antes e depois do LLM, e também nas chamadas de ferramentas. *Guardrails* não devem ser apenas instruções no *system prompt*; controles determinísticos e classificadores externos são necessários para fluxos críticos.

Camada de observabilidade: coleta logs, métricas, rastreamento distribuído, ativações de *guardrails*, versões de *prompts*, identificadores de modelo, consumo de *tokens*, TTFT, latência P95/P99 e resultados de avaliação contínua. Logs devem respeitar minimização, pseudonimização e retenção compatível com LGPD.

Os principais pontos de falha são: entrada maliciosa não bloqueada, conteúdo externo tratado como instrução, *prompt* incompatível com nova versão do modelo, saída sem validação, ferramenta chamada com argumentos inválidos, vazamento de dados por logs e ausência de *fallback* quando a confiança é baixa. Os principais limites de segurança ficam no API Gateway, no orquestrador, na validação de ferramentas, no servidor de inferência e na política de logs.

4.6 Dinâmica de execução e ciclos de controle

Além da arquitetura estática, sistemas LLM em produção devem explicitar o fluxo dinâmico de execução. A interação real raramente é um pipeline linear; normalmente opera como ciclo de controle com classificação, decisão, execução, validação e recuperação de falhas.

1. Receber requisição e autenticar usuário
2. Classificar intenção, risco e escopo de autorização
3. Recuperar contexto permitido e montar *prompt* versionado
4. Decidir rota: LLM, RAG, ferramenta, agente, humano ou recusa
5. Executar chamada com limites de *tokens*, tempo e custo
6. Validar saída, fundamentação, *schema* e políticas de segurança
7. Retentar (*retry*), degradar, escalar ou finalizar
8. Registrar decisão, artefatos, métricas e eventos de auditoria

Esse ciclo deve ser idempotente sempre que possível. Em especial, operações com efeito

transacional não devem ser repetidas apenas porque o LLM falhou ou retornou resposta inválida. Tentativas automáticas devem distinguir falhas recuperáveis, como erro de *parsing* ou *timeout* temporário, de falhas de política, como falta de autorização, classificação de risco alto ou baixa confiança na fundamentação.

Pontos de controle recomendados: antes do LLM, classificar intenção, sensibilidade dos dados e risco de injeção de *prompt*; durante a execução, controlar orçamento de *tokens*, tempo, ferramentas e *retries*; depois do LLM, validar *schema*, política, fundamentação, toxicidade, dados pessoais e consistência com o contexto autorizado.

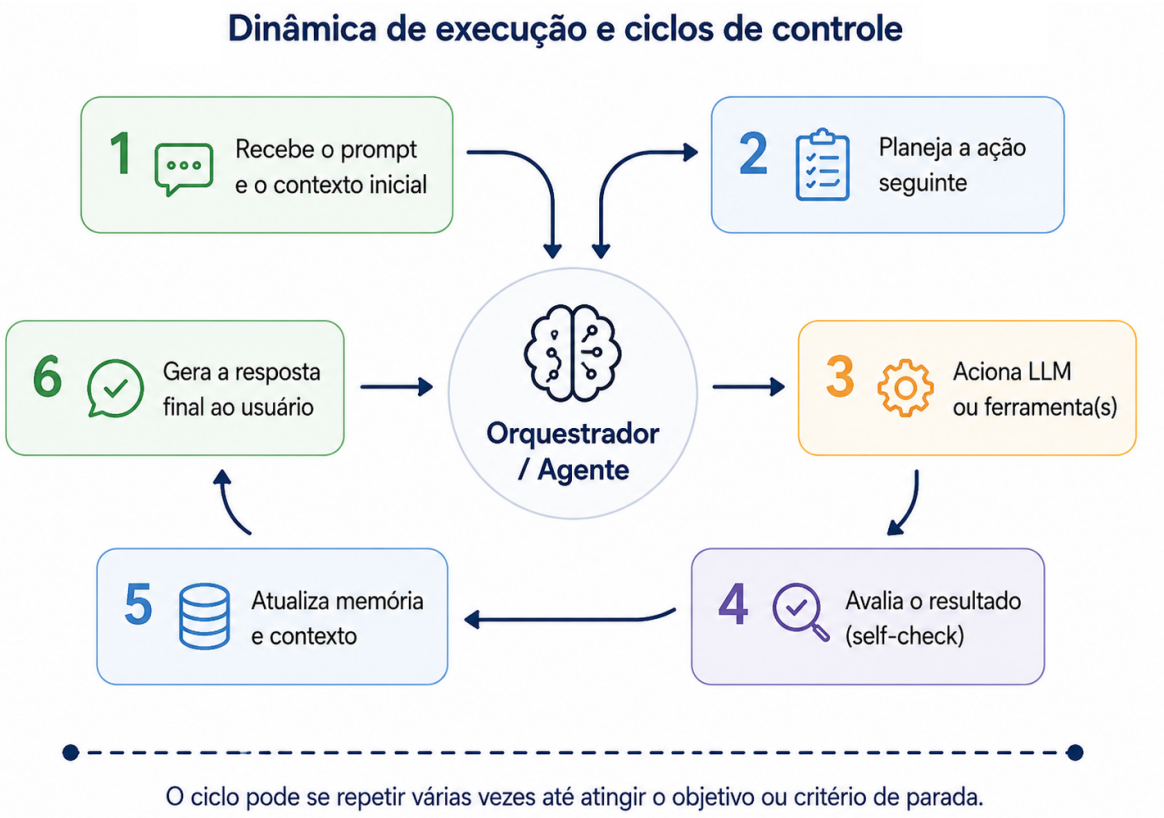

Figura 7 - Dinâmica de execução e ciclos de controle do orquestrador

A imagem reforça que sistemas LLM em produção operam como ciclos de controle, e não apenas como pipelines lineares. As etapas de planejamento, acionamento de LLM ou ferramentas, avaliação, atualização de contexto e geração final se conectam aos critérios de *retry, fallback* e parada descritos na seção.

4.6 Políticas de decisão do orquestrador

O orquestrador deve implementar políticas explícitas para escolher modelo, contexto, ferramentas, *fallback* e escalação humana. Essas políticas não devem ficar implícitas no *prompt*, pois afetam custo, segurança, rastreabilidade e qualidade do serviço.

Políticas baseadas em regras: usam limiares determinísticos, como canal de atendimento, perfil do usuário, classificação de risco, volume de *tokens*, tipo de tarefa e criticidade do ser-

viço. São mais auditáveis e adequadas para decisões de autorização, bloqueio, recusa, uso de ferramenta sensível e escalação humana.

Políticas aprendidas: usam classificadores ou modelos para estimar intenção, dificuldade, risco, probabilidade de sucesso, necessidade de RAG ou necessidade de ferramenta. Podem melhorar eficiência, mas exigem *dataset* de validação, monitoramento de *drift*, explicabilidade operacional e *fallback* determinístico.

Políticas híbridas: combinam regras obrigatórias com estimativas aprendidas. Uma regra pode proibir ações fora do escopo do usuário, enquanto um classificador escolhe entre modelo pequeno, modelo avançado, RAG ou revisão humana. Esse desenho costuma ser o mais apropriado para produção, pois preserva governança em decisões críticas e usa modelos para otimizar custo e qualidade.

```
if risco_alto or acao_sensivel:
    rota = "revisao_humana"
elif precisa_dado_atualizado:
    rota = "RAG_ou_ferramenta"
elif complexidade <= limiar and baixo_risco:
    rota = "modelo_menor"
else:
    rota = "modelo_avancado_com_validacao"
```

A decisão deve registrar: política aplicada, versão da política, *features* usadas, modelo escolhido, razão do *fallback*, confiança dos classificadores, ferramentas permitidas e resultado das validações. Esse registro é essencial para auditoria, depuração, contestação de decisões e análise de custo.

5. Técnicas, arquiteturas e métodos

Esta seção detalha os fundamentos técnicos e conceituais que sustentam sistemas com LLMs, fornecendo orientação para tomada de decisão arquitetural, seleção de componentes e avaliação de conformidade técnica.

5.1 Prompt engineering

Prompt engineering é o conjunto de técnicas para formular instruções que direcionam o comportamento do LLM sem modificar seus pesos. Representa a forma mais direta e econômica de adaptar o modelo a tarefas específicas.

***Zero-shot*:** instrução direta sem exemplos. O modelo aplica seu conhecimento paramétrico para executar a tarefa descrita. Adequado quando a tarefa é suficientemente genérica e o modelo já foi treinado em contextos similares. Exemplo: "Classifique o sentimento do texto a seguir como positivo, negativo ou neutro: [texto]".

***Few-shot*:** inclusão de dois a cinco exemplos de entrada-saída antes da consulta real. Calibra

o formato, o estilo e o nível de detalhe esperado sem exigir ajuste fino. Particularmente eficaz em tarefas de extração estruturada e classificação com categorias institucionais específicas.

***Chain-of-Thought* (CoT):** instrução explícita para que o modelo produza uma simulação de inferência passo a passo antes de concluir. Pode melhorar a acurácia em tarefas de lógica, matemática e análise jurídica, mas não deve ser descrito como raciocínio humano: o modelo continua executando previsão estatística de próximos *tokens*. A exposição integral da cadeia ao usuário nem sempre é desejável em produção, pois pode revelar heurísticas internas, aumentar custo e facilitar ataques. A variante *zero-shot CoT* utiliza instrução curta, como "pense passo a passo", sem exemplos adicionais. Técnicas relacionadas incluem *self-consistency CoT*, em que múltiplas cadeias são amostradas e agregadas, e *Tree of Thought*, que explora caminhos alternativos de solução. Modelos recentes também podem oferecer inferência estendida nativa, controlada por parâmetros ou modos do fornecedor, indo além do CoT clássico induzido por *prompt*.

Boas práticas para contexto governamental: instruções devem ser explícitas quanto ao papel do modelo ("Você é um assistente institucional..."), ao formato de saída esperado e às restrições de conteúdo. O *system prompt* deve ser testado contra casos adversariais antes de entrar em produção e versionado junto ao código da aplicação.

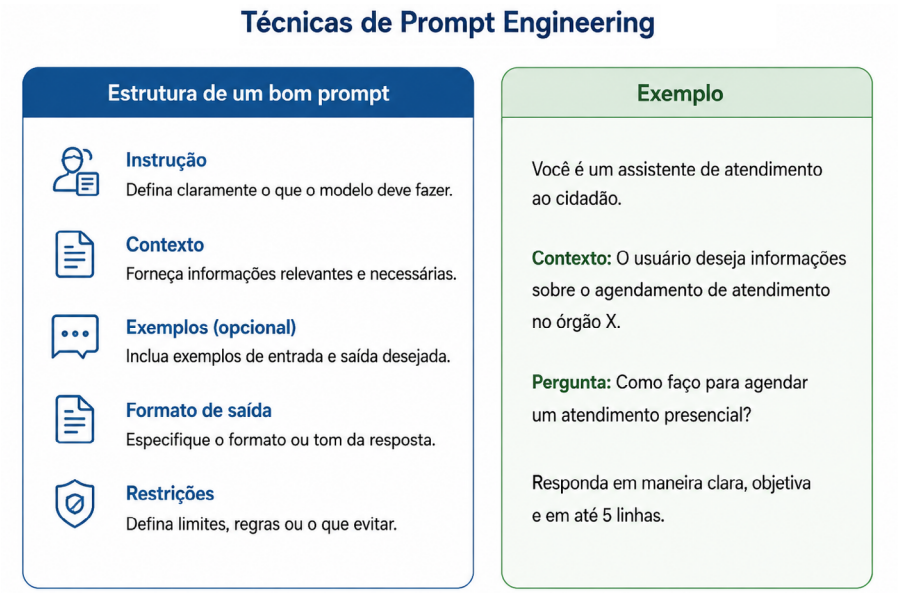

Figura 8 – Estrutura de prompt engineeging

A figura organiza os elementos essenciais de um *prompt*: instrução, contexto, exemplos, formato de saída e restrições. O exemplo de atendimento ao cidadão demonstra como essas partes podem ser combinadas para orientar comportamento, escopo e forma da resposta sem alterar os pesos do modelo.

5.2 Parâmetros de decodificação

Os parâmetros de decodificação controlam como o modelo seleciona tokens na geração. Sua configuração adequada é crítica para equilibrar precisão, diversidade e reprodutibilidade conforme o caso de uso.

Temperatura: reescala os *logits* antes do *softmax* e, portanto, altera a entropia da distribuição de *tokens* amostrada. Valores próximos de zero concentram a probabilidade nos *tokens* mais prováveis e tornam a saída mais determinística; valores acima de 1,0 aumentam a entropia e a diversidade da amostragem. "Criatividade" é um efeito comportamental observado, não uma propriedade técnica do parâmetro. Para sistemas governamentais que exigem precisão factual e reprodutibilidade, recomenda-se temperatura entre 0,0 e 0,3.

Formalmente, dado um *logit* z_i para o *token* i e temperatura T:

$$P_i = \frac{e^{z_i/T}}{\sum_j e^{z_j/T}}$$

Quando $T \to 0$, a distribuição se concentra no *token* de maior *logit*. Quando T aumenta, a distribuição se torna mais plana e *tokens* menos prováveis passam a ter maior chance de seleção.

***Top-K*:** restringe a amostragem aos K *tokens* mais prováveis em cada passo, descartando opções de baixa probabilidade. Reduz o risco de *tokens* incoerentes sem eliminar variabilidade útil.

***Top-p* (*nucleus sampling*):** seleciona o menor conjunto de *tokens* cuja probabilidade acumulada atinge o limiar p (ex.: 0,9). Adapta dinamicamente o número de candidatos conforme a entropia da distribuição, sendo mais robusto que *Top-K* em distribuições assimétricas.

Se os *tokens* forem ordenados por probabilidade decrescente, o conjunto de amostragem é o menor conjunto S tal que:

$$\sum_{i \in S} P_i \geq p$$

Após a seleção de S, as probabilidades são renormalizadas e a amostragem ocorre apenas dentro desse núcleo.

Implicações práticas: temperatura baixa + *top-p* conservador (0,85–0,90) é configuração recomendada para aplicações de atendimento ao cidadão e geração de documentos institucionais, onde consistência e fidelidade ao contexto superam diversidade estilística.

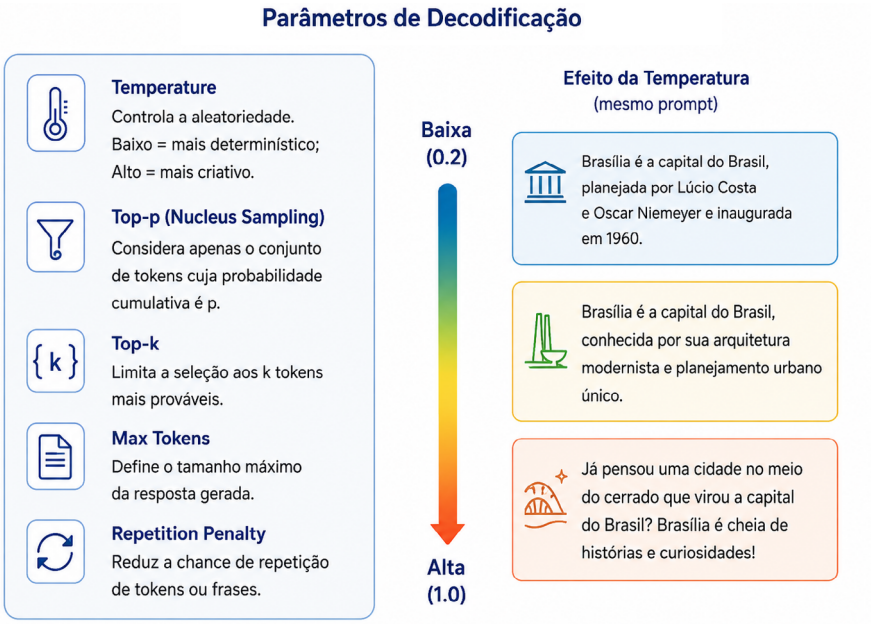

Figura 9 - Parâmetros de decodificação e efeito da temperatura

A imagem sintetiza os principais parâmetros que afetam a geração, incluindo temperatura, *top-p, top-k, max tokens* e penalidade de repetição. Os exemplos à direita mostram como a mesma pergunta pode produzir respostas mais determinísticas ou mais livres conforme a temperatura aumenta.

5.3 Fine-tuning e alinhamento

Fine-tuning é o processo de ajustar os pesos do modelo-base com dados específicos do domínio ou da tarefa. Justifica-se quando *prompt engineering* não atinge o desempenho necessário ou quando o custo de incluir exemplos repetidamente no *prompt* é economicamente proibitivo.

SFT (*Supervised Fine-Tuning*): ajuste com pares instrução-resposta curados por especialistas de domínio. Adapta o estilo, o vocabulário técnico e o formato de saída do modelo. Requer *dataset* de qualidade (centenas a milhares de exemplos) e infraestrutura de treinamento compatível com o tamanho do modelo.

PEFT/LoRA (*Parameter-Efficient Fine-Tuning / Low-Rank Adaptation*): técnicas que ajustam apenas uma pequena fração dos parâmetros do modelo (matrizes de baixo posto adicionadas às camadas de atenção), reduzindo drasticamente o custo computacional e de armazenamento. Viável em infraestrutura de médio porte sem perda significativa de qualidade em relação ao ajuste fino completo.

Quantização e QLoRA: quantização reduz a precisão numérica dos pesos e ativações, por exemplo para INT8, INT4 ou formatos de distribuição como GGUF e variantes GPTQ/AWQ, permitindo executar modelos em hardware de menor custo. A redução de memória pode vir acompanhada de perda de qualidade, maior sensibilidade a configuração e necessidade de validação específica por tarefa. QLoRA combina LoRA com carregamento quantizado do modelo-base, sendo uma técnica prática para *fine-tuning* eficiente em ambientes com GPUs limitadas.

RLHF e DPO (*Reinforcement Learning from Human Feedback / Direct Preference Optimization*): técnicas de alinhamento que treinam o modelo para preferir respostas avaliadas como mais úteis, seguras e corretas por avaliadores humanos. RLHF utiliza um modelo recompensa intermediário; DPO otimiza diretamente sobre pares de preferências, sendo mais estável e menos custoso. Ambas são relevantes quando o comportamento padrão do modelo-base é inadequado para o contexto institucional.

Quando justifica-se *fine-tuning*: quando a tarefa exige terminologia institucional específica não coberta pelo modelo-base; quando a latência impede *prompts* longos com muitos exemplos; quando há volume suficiente de dados curados e orçamento de treinamento disponível. Caso contrário, *prompt engineering* bem estruturado oferece retorno mais rápido com menor risco.

***Instruction tuning* vs. *domain tuning*:** *instruction tuning* adapta o modelo a seguir formatos e instruções específicas; *domain tuning* adapta vocabulário, estilo e padrões de um domínio. O primeiro costuma exigir pares instrução-resposta; o segundo pode usar corpus institucional, mas aumenta risco de memorizar documentos e de reduzir capacidade geral se mal calibrado.

Falhas comuns de *fine-tuning*: *catastrophic forgetting*, quando o modelo perde capacidades gerais; *overfitting* em datasets pequenos; vazamento de avaliação, quando exemplos de teste entram no treinamento; e regressões de segurança, quando o ajuste melhora uma tarefa mas enfraquece recusas, privacidade ou aderência a políticas. Todo *fine-tuning* deve ter conjunto de avaliação separado, controle de versão de dados, comparação contra o modelo base e testes de segurança pós-ajuste.

5.4 Gerenciamento de janela de contexto

A janela de contexto é o limite máximo de *tokens* que o modelo processa em uma única inferência, abrangendo *system prompt*, histórico de conversa e consulta do usuário. Modelos atuais variam de 4K a 200K+ *tokens* dependendo da arquitetura e do fornecedor.

Estratégias para documentos longos: quando o conteúdo excede a janela, é necessário adotar abordagens de processamento fragmentado:

- *Map-reduce summarization*: o documento é dividido em segmentos que são resumidos individualmente (*map*), e os resumos são consolidados em uma resposta final (*reduce*);
- *Sliding window*: janela deslizante com sobreposição para preservar contexto entre segmentos consecutivos;
- Sumarização hierárquica: resumos de primeiro nível são resumidos novamente até caber na janela final.

***FinOps* de *tokens*:** o custo de APIs comerciais é proporcional ao total de **tokens** de entrada e saída. Práticas recomendadas incluem: medir e registrar o consumo médio por tipo de consulta; definir limites de *tokens* por conversa; monitorar custos em *dashboards* integrados ao ciclo MLOps/FinOps do projeto; e avaliar modelos de menor custo para tarefas menos exigentes (roteamento por complexidade).

Uma estimativa mínima de custo por chamada deve considerar:

$$C = T_{in} \times P_{in} + T_{out} \times P_{out}$$

Em que T_{in} e T_{out} são *tokens* de entrada e saída, e P_{in} e P_{out} são os preços por *token* de entrada e saída. Estratégias de redução de custo incluem compressão de *prompt*, cache de respostas ou trechos estáveis, roteamento entre modelos por complexidade, limites de saída, sumarização de histórico e reuso de contexto quando suportado pelo provedor.

O custo total também deve incluir componentes que não aparecem na fórmula simples de chamada ao LLM. Em arquiteturas com RAG, há custo de geração de *embeddings*, armazenamento vetorial, reindexação, consultas ao índice e eventual *reranking*. Em arquiteturas auto-hospedadas, há custo de GPU/hora, reserva de capacidade para picos, ociosidade, energia, observabilidade, engenharia de plataforma e atualização de modelos. Em sistemas "agentivos", o custo pode crescer de forma multiplicativa, pois uma única solicitação do usuário pode gerar várias chamadas ao LLM, ferramentas, consultas vetoriais e validações intermediárias.

Para controle de FinOps, cada rota do orquestrador deve declarar orçamento máximo por interação, incluindo T_{in}, T_{out}, número máximo de chamadas, número máximo de recuperações vetoriais, tempo de execução e custo estimado. Alertas devem considerar custo por usuário, custo por sessão, custo por tarefa concluída e anomalias como loops "agentivos", *prompts* excessivamente longos e aumento súbito de *tokens* de saída.

5.5 Emeddings e representação vetorial

Embeddings são vetores numéricos que representam textos, trechos documentais, consultas ou respostas em um espaço semântico. Textos com sentido semelhante tendem a ficar próximos nesse espaço, permitindo busca semântica, agrupamento, deduplicação, avaliação de relevância e métricas como BERTScore.

A similaridade mais comum é a similaridade do cosseno entre dois vetores a e b:

$$cos(a, b) = \frac{a \cdot b}{\|a\| \|b\|}$$

***Trade-offs* principais:** dimensões maiores podem capturar mais nuance semântica, mas aumentam custo de armazenamento, memória e busca. Modelos de *embedding* diferentes não são diretamente comparáveis: trocar o modelo exige reindexação e revalidação. *Embeddings* também podem codificar dados sensíveis; portanto, vetores, índices e logs de busca devem ser tratados como ativos de dados protegidos.

Índices vetoriais e ANN: em bases pequenas, busca exata pode ser suficiente. Em bases grandes, usa-se busca aproximada de vizinhos mais próximos (*Approximate Nearest Neighbors* – ANN), com estruturas como HNSW, IVF ou FAISS. Essas técnicas reduzem latência e custo,

mas introduzem trade-off entre *recall*, memória, tempo de indexação e tempo de consulta. A configuração do índice deve ser validada com consultas reais do domínio, medindo *recall@k*, latência P95/P99 e impacto na qualidade final das respostas.

Estratégias de *chunking*: documentos devem ser divididos em unidades recuperáveis com tamanho, sobreposição e metadados adequados. *Chunks* muito pequenos perdem contexto; *chunks* muito grandes reduzem precisão, aumentam custo e podem carregar trechos irrelevantes para o LLM. Estratégias comuns incluem divisão por seção, parágrafo, janela com sobreposição, hierarquia documento-seção-trecho e *chunking* semântico. Para documentos normativos, preservar títulos, artigos, incisos, datas, fonte e versão é tão importante quanto o texto do trecho.

***Drift* semântico:** mudanças no corpus, no vocabulário do domínio, no modelo de *embedding* ou no perfil de consulta podem reduzir a qualidade de similaridade ao longo do tempo. Monitorar distribuição de consultas, taxa de reformulação, relevância avaliada por humanos e exemplos de falsos positivos/falsos negativos é necessário para manter qualidade em produção.

5.6 Memória conversacional

Sistemas conversacionais dependem de estado. **Memória de curto prazo** é o histórico mantido na janela de contexto da conversa; melhora continuidade, mas aumenta custo, latência e risco de carregar instruções maliciosas de turnos anteriores. **Memória de longo prazo** usa bancos relacionais, documentos, perfis ou bases vetoriais para recuperar informações persistentes; melhora personalização, mas eleva riscos de privacidade, autorização e retenção indevida.

O estado de sessão deve ter identificador, escopo, tempo de vida, política de expiração, segregação por usuário e registro de consentimento quando envolver dados pessoais. A aplicação deve distinguir memória informativa de instrução: conteúdo salvo de conversas anteriores não deve ter autoridade para alterar políticas do sistema. Em serviços públicos, memória persistente deve ser justificada por finalidade, minimizada e auditável.

5.7 Serving de modelos e infraestrutura de inferência

Em modelos auto-hospedados, a camada de serving deve ser tratada como decisão arquitetural explícita. Frameworks como vLLM, *Hugging Face Text Generation Inference* e NVIDIA *Triton Inference* Server oferecem recursos distintos de paralelismo, *batching*, *streaming*, gestão de memória, cache de atenção e observabilidade. A escolha deve considerar TTFT (*Time to First Token*), tokens por segundo, latência P95/P99, isolamento entre usuários, suporte a quantização, auditoria de logs e controles de rede.

Para cargas de atendimento ao cidadão, recomenda-se validar a infraestrutura com testes de carga realistas, incluindo concorrência, consultas longas, respostas em *streaming*, limites de contexto e cenários de degradação. O servidor de inferência deve expor métricas operacionais para o ambiente de MLOps/FinOps e possuir procedimento de *rollback* para troca de modelo, configuração de quantização ou parâmetros de *serving*.

***Trade-offs de serving*:** *streaming* reduz a percepção de latência, mas pode exibir texto antes da validação final de saída; respostas não transmitidas permitem validação completa antes da exibição, mas pioram TTFT percebido. O *Batching* dinâmico melhora *throughput*, mas pode aumentar latência de cauda. Reuso de KV cache reduz custo de prefixos repetidos, mas exige controle cuidadoso de isolamento entre usuários. *Speculative decoding* pode acelerar geração usando um modelo auxiliar menor, mas adiciona complexidade e requer validação de compatibilidade com o modelo principal.

5.8 Uso de ferramentas e function calling

Sistemas LLM podem ser conectados a ferramentas externas, como APIs governamentais, bases de dados, mecanismos de busca, calculadoras, serviços de protocolo ou fluxos automatizados. Essa capacidade, frequentemente chamada de *function calling* ou *tool use*, permite que o modelo selecione uma função, preencha argumentos estruturados e utilize o resultado na resposta final.

O uso de ferramentas aumenta a utilidade do sistema, mas também amplia a superfície de risco. Toda função deve ter contrato de entrada e saída estruturado, validação de argumentos fora do modelo, autorização por escopo mínimo, limites de taxa, trilha de auditoria e política clara de quais ações podem ser executadas automaticamente. Funções com efeito transacional, alteração cadastral, envio de comunicação oficial ou impacto sobre direitos de cidadãos exigem confirmação humana ou controle equivalente antes da execução.

Outputs estruturados: quando o sistema precisa produzir JSON, campos de formulário, comandos ou argumentos de função, deve-se usar esquema explícito, preferencialmente JSON *Schema* ou mecanismo equivalente do provedor. A aplicação deve validar o objeto gerado com *parser* estrito, rejeitar campos inesperados, aplicar tipos e faixas permitidas e tratar falhas de *parsing* com *retry* controlado ou escalação. Saída estruturada reduz ambiguidade, mas não substitui validação de negócio.

***Prompt templates* versionados:** modelos de *prompt* devem ser parametrizados, versionados e testados como código. Cada *template* deve declarar variáveis aceitas, fontes de dados usadas, política de escape/delimitação, versão do modelo compatível e conjunto de testes de regressão. Mudança em *template* deve passar por revisão e avaliação automatizada antes de produção.

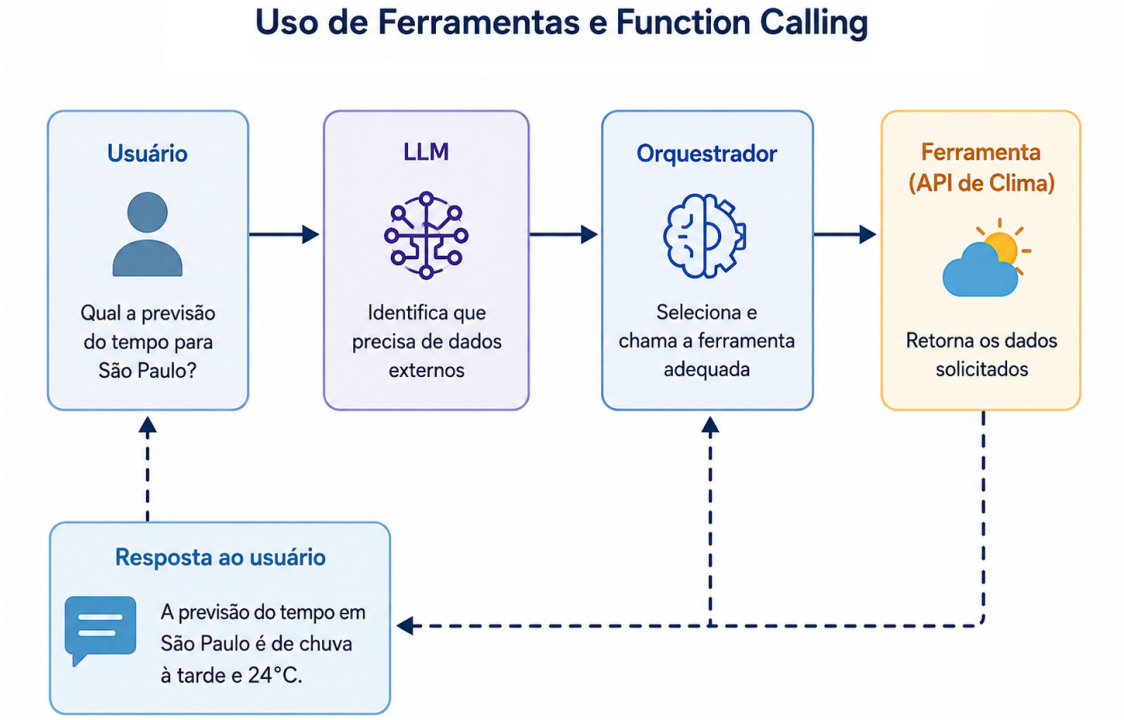

Figura 10 - Uso de ferramentas e ***function calling***

A figura exemplifica o uso de uma ferramenta externa quando o modelo identifica necessidade de dados que não estão no *prompt* nem no conhecimento paramétrico. O orquestrador seleciona e chama a ferramenta apropriada, enquanto a resposta final deve ser montada a partir dos dados retornados e validada antes de chegar ao usuário.

5.9 Sistemas "agentivos" e orquestração multi-etapas

Sistemas "agentivos" combinam LLM, ferramentas, memória e ciclos de planejamento/execução para resolver tarefas em múltiplas etapas. Um agente pode decompor uma solicitação, escolher ferramentas, observar resultados, revisar plano e produzir resposta final. Essa abordagem é útil para fluxos administrativos complexos, mas aumenta risco de agência excessiva, execução indevida, loops, consumo imprevisível de *tokens* e dificuldade de auditoria.

Para uso governamental, agentes devem operar com orçamento máximo de passos, limite de tempo, lista permitida de ferramentas, autorização por escopo, política de confirmação humana para ações sensíveis e trilha de auditoria por etapa. O plano gerado pelo modelo deve ser tratado como hipótese operacional, não como decisão autorizada. A orquestração deve ser determinística sempre que possível, reservando ao modelo apenas as partes de interpretação, classificação ou geração que exigem linguagem natural.

Controles práticos de execução: agentes devem ter limites explícitos de passos, *tokens*, tempo, chamadas de ferramenta e custo por tarefa. Também devem possuir critérios de parada, detecção de repetição de ações, validação de progresso e política de falha de plano. Sem

esses controles, o sistema pode entrar em loops, repetir consultas sem ganho, consumir orçamento excessivo ou executar ações incompatíveis com o objetivo original.

max_steps = 5
max_llm_calls = 8
max_tokens_total = 12000
timeout_seconds = 60
max_tool_calls_per_tool = 2
stop_if_no_new_evidence = true
escalate_if_plan_fails = true

Falha de plano: ocorre quando o agente não consegue avançar apesar de múltiplas etapas, quando a ferramenta retorna dados insuficientes, quando há conflito entre objetivos ou quando a validação reprova repetidamente a saída. Nesses casos, o comportamento correto é encerrar com explicação limitada, solicitar informação adicional, degradar para fluxo determinístico ou escalar para humano, e não continuar tentando indefinidamente.

5.10 Arquitetura de guardrails

Guardrails são controles técnicos e processuais que restringem entradas, saídas e ações do sistema LLM de acordo com políticas de segurança, privacidade, qualidade e conformidade. Uma arquitetura mínima deve combinar:

(i) *guardrails* de entrada, como classificação de intenção, detecção de injeção direta e indireta de *prompt* e mascaramento de dados pessoais;

(ii) *guardrails* de contexto, como separação entre dados confiáveis e conteúdo externo não confiável;

(iii) *guardrails* de saída, como classificadores de toxicidade, vazamento de dados sensíveis, fundamentação e conformidade; e

(iv) *guardrails* de ação, que validam chamadas de ferramentas antes da execução.

Implementações podem combinar modelos classificadores leves, regras determinísticas, NLI para verificação de suporte textual, filtros de PII, soluções especializadas como NeMo *Guardrails* e modelos de segurança como *LlamaGuard*. O desenho adotado deve ser documentado com limiares, taxa esperada de falsos positivos, procedimento de revisão humana e estratégia de atualização.

Tipos de guardrail: *guardrails* determinísticos aplicam regras, *parsers*, listas de bloqueio, validação de *schema* e limites de taxa; são previsíveis e auditáveis, mas pouco flexíveis. *Guardrails* baseados em ML classificam intenção, toxicidade, injeção, sensibilidade e fundamentação; cobrem casos variados, mas introduzem latência, falsos positivos e falsos negativos. *Guardrails in-context* usam instruções no *prompt*; são baratos, mas frágeis contra bypass e não devem ser a única camada de defesa.

Posicionamento e resposta: controles *pre-LLM* bloqueiam ou saneiam entradas; controles *post-LLM* validam respostas antes da exibição; controles de ação validam chamadas de ferramenta. A política pode usar *hard blocking* (bloqueio), *soft blocking* (resposta segura alternativa),

mascaramento, pedido de esclarecimento, execução em modo sombra ou revisão humana. Cascatas de classificação podem reduzir custo: regra simples primeiro, classificador leve em seguida e modelo avaliador apenas em casos ambíguos.

Falhas esperadas: ataques *multi-turn* podem distribuir a instrução maliciosa em vários turnos; ataques de *bypass* podem usar codificação, idiomas mistos, role-play ou fragmentação de comandos; conteúdo externo pode induzir execução de ferramenta. Por isso, logs de ativações, amostras de falsos negativos, testes adversariais e atualização periódica dos classificadores são parte do controle, não atividades opcionais.

Figura 11 - Arquitetura de ***guardrails*** para sistemas LLM

A imagem consolida os principais pontos de proteção: filtros de entrada, filtros de saída, políticas e regras, monitoramento contínuo e análise de uso. Ela complementa a seção ao mostrar que *guardrails* devem atuar em múltiplas camadas, não apenas como instruções no system *prompt*.

5.11 Red teaming adversarial

O *red teaming* deve ser planejado como atividade técnica reprodutível, e não apenas como exploração informal. O plano mínimo deve cobrir *jailbreak* por *role-play*, injeção em campos estruturados, injeção indireta via documentos ou páginas externas, extração de *system prompt*, escalada de contexto em conversas longas, vazamento de dados pessoais, uso indevido de ferramentas, negação de serviço por contexto excessivo e geração de desinformação.

Os resultados devem registrar vetor testado, exemplo de entrada, resposta observada, severidade, controle que falhou, evidência, decisão de remediação e teste de regressão associado. A amostragem deve incluir consultas sintéticas e casos reais anonimizados, com prioridade para fluxos de alto impacto e para integrações que executem ações externas.

6. Avaliação e métricas

A avaliação de qualidade em sistemas LLM é multidimensional, envolvendo a qualidade das respostas geradas, a aderência ao *prompt* e às instruções do sistema, a ausência de conteúdo nocivo e a satisfação dos usuários. Esta seção estabelece métricas de referência, orientações de interpretação, valores de referência e processos de avaliação para sistemas LLM no setor público.

Ressalva metodológica sobre valores de referência

A adoção de exemplos internos como valores de referência exige cautela metodológica. Conforme a literatura recente, a avaliação de sistemas LLM é estritamente dependente de contexto, variando conforme o domínio, *corpus*, tarefa, configuração do *prompt*, conjunto de teste e métricas utilizados.

Dessa forma, inexistem limiares universais estáveis; o desempenho deve ser interpretado exclusivamente dentro das especificidades deste cenário, não servindo como padrão absoluto para futuras implementações. Os limiares efetivamente adotados devem ser formalmente registrados no ETP ou no Plano de Experimentação do projeto e revisados com dados de produção.

6.1 Métricas de geração de texto

Métricas de geração avaliam qualidade das respostas produzidas pelo LLM condicionadas ao contexto recuperado.

Distinção conceitual: *groundedness* mede se a resposta está suportada pelo contexto fornecido; *factual accuracy* mede se as afirmações são verdadeiras no mundo; e alucinação é geração de afirmação falsa ou sem suporte apresentada como fato. Em LLM puro, *groundedness* só é diretamente aplicável quando há contexto explícito no *prompt*. Uma resposta pode ser factual, mas não fundamentada no contexto; também pode estar fundamentada em um contexto fornecido pelo usuário que esteja incorreto. Por isso, sistemas de alto impacto devem combinar fundamentação, verificação factual por fonte confiável e revisão humana.

6.1.1 Aderência ao ***Prompt / Groundedness*** (Fundamentação)

Definição: Grau em que a resposta gerada é **fundamentada** no conteúdo fornecido pelo usuário no próprio *prompt* (documentos inseridos, histórico de conversa, instruções do sistema), sem acrescentar informação não presente nesse contexto.

Interpretação: Aderência = 1,0: **Todas** as afirmações na resposta têm suporte no conteúdo fornecido no *prompt*. Aderência = 0,8: 80% das afirmações têm suporte; 20% provêm do conhecimento paramétrico do modelo (possível alucinação). Aderência = 0,5: metade das afirmações não são fundamentadas no *prompt*.

Por que é crítica: Detecta **alucinações** – quando o LLM gera informação plausível mas factualmente incorreta ou não suportada pelo conteúdo fornecido. Em LLM puro sem contexto explícito, a métrica não prova verdade factual; ela mede apenas aderência ao que foi fornecido na conversa. Sem base externa confiável, o modelo pode "preencher lacunas" com conhecimento paramétrico desatualizado ou incorreto. Em contexto governamental, alucinações podem resultar em orientação incorreta com consequências legais, operacionais ou para cidadãos.

Como avaliar:

Método automático: Frameworks de avaliação de LLMs (ex.: LLM-as-a-Judge) analisam sobreposição semântica entre afirmações na resposta e conteúdo do *prompt*: (1) decompor resposta em afirmações individuais; (2) para cada afirmação, verificar se há suporte no *prompt*; (3) calcular proporção de afirmações suportadas.

Método manual: Especialistas leem resposta e *prompt*, verificando se cada afirmação tem evidência no conteúdo fornecido.

Exemplos internos de valores de referência:

- **Sistemas críticos** (jurídico, saúde, decisões com impacto significativo): Aderência > 0,85
- **Sistemas gerais:** Aderência > 0,75
- **Abaixo de 0,7:** Ação corretiva necessária (alto risco de desinformação)

Como melhorar: Refinar *system prompt* com instruções explícitas: "Responda APENAS com base no conteúdo fornecido. Se a informação não está disponível, declare claramente que não sabe". Usar modelos com melhor seguimento de instruções. Implementar validação pós-geração para detectar e filtrar respostas não fundamentadas. Reduzir temperatura de geração (menor entropia de amostragem, maior consistência em relação ao contexto).

6.1.2 Answer Relevancy (Relevância da Resposta)

Definição: Grau em que a resposta aborda **diretamente** a consulta do usuário.

Interpretação: Resposta pode ser perfeitamente fundamentada (*faithful*) mas não responder a pergunta feita. Exemplo: Consulta: "Qual é o prazo para aposentadoria? ". Resposta fundamentada mas irrelevante: "A aposentadoria é um direito dos servidores públicos previsto no Art. X...". Resposta relevante: "O prazo para solicitação de aposentadoria é de 30 dias antes da data desejada, conforme Norma RH-001. "

Como avaliar: Similaridade semântica entre consulta e resposta (*embeddings*). Avaliação de especialistas: A resposta aborda a pergunta? Análise de feedback de usuários.

Trade-off: Respostas muito curtas e diretas têm alta relevância mas podem omitir contexto importante. Respostas completas e contextualizadas podem ter relevância ligeiramente menor mas são mais úteis.

Valor mínimo: *Answer Relevancy* > 0.7.

6.1.3 Perplexidade (Perplexity)

Definição: Medida de quão bem o modelo prevê uma sequência de texto; reflete a fluidez e coerência linguística da geração. Valores menores indicam texto mais previsível e fluente.

Para uma sequência de N *tokens* com probabilidades atribuídas pelo modelo, a perplexidade pode ser expressa como:

$$PP = exp\left(-\frac{1}{N}\sum_{i=1}^{N} logP\left(x_i\right)\right)$$

Uso: Avaliação do modelo base durante seleção e após *fine-tuning*. Não é aplicável como métrica primária de qualidade em produção, pois texto fluente pode ainda conter alucinações. Complementa avaliações de aderência ao *prompt*.

Limitação: Perplexidade baixa não implica correção factual; mede apenas adequação estatística à distribuição de linguagem do treinamento. Em produção, há frequentemente *mismatch* entre a distribuição de treinamento e a distribuição real da tarefa, do domínio institucional e dos usuários. Um texto pode ser provável para o modelo e ainda assim estar errado, inseguro ou fora da política.

6.1.4 BLEU / ROUGE (Tarefas com Referência)

Definição: Métricas de sobreposição superficial entre texto gerado e texto de referência (tradução esperada, resumo esperado). BLEU mede precisão de n-gramas; ROUGE mede revocação de n-gramas.

Quando usar: Tarefas com saída bem definida e referência disponível: tradução automática, sumarização extrativa, geração de respostas factuais com gabarito.

Limitações: São métricas de superfície e não capturam correção semântica ou factual. Dois textos podem ter BLEU alto mas significados divergentes. Não devem ser usadas como única métrica de qualidade.

Exemplos internos de valores de referência: BLEU > 0,3 para tradução de qualidade aceitável; ROUGE-L > 0,4 para sumarização com boa cobertura de conteúdo essencial.

6.1.5 BERTScore e métricas semânticas

Definição: BERTScore compara texto gerado e referência por similaridade entre *embeddings* contextuais, capturando equivalência semântica melhor do que métricas baseadas apenas em sobreposição de n-gramas.

Quando usar: Tarefas com referência textual em que paráfrases corretas são aceitáveis, como sumarização abstrativa, respostas orientadas por gabarito, tradução com variação lexical e reescrita institucional.

Limitações: A métrica depende do modelo de *embedding* utilizado, pode superestimar textos semanticamente próximos mas factualmente incorretos e não substitui avaliação de fundamentação em casos de alto risco. Deve ser usada em conjunto com revisão humana, aderência ao *prompt* e métricas de segurança.t

6.1.6 Toxicidade e Viés (Toxicity/Bias Score)

Definição: Proporção de respostas que contêm conteúdo nocivo (linguagem ofensiva, discriminatória, violenta) ou que reproduzem vieses presentes nos dados de pré-treinamento do modelo.

Por que é crítica: Sistemas LLM podem reproduzir preconceitos de seus dados de treinamento. Em contexto governamental, respostas enviesadas ou discriminatórias expõem a instituição a responsabilização legal, violam o princípio constitucional de isonomia e comprometem a confiança da sociedade no serviço público.

Como avaliar: Classificadores automáticos de toxicidade (ex.: Perspective API, modelos NLI treinados para detecção de conteúdo nocivo) aplicados sobre amostras de interações de produção. Avaliação humana periódica por especialistas de equidade (*fairness auditors*).

Exemplos internos de valores de referência:

- **Taxa de toxicidade:** < 0,5% das respostas em sistemas de atendimento ao cidadão
- **Abaixo de 0,5%:** Sistema operando dentro do limiar aceitável (monitoramento contínuo obrigatório)
- **Acima de 1%:** Ação corretiva obrigatória (revisão de guardrails, *system prompt* e fine-tuning)

Como melhorar: Reforçar instruções contra conteúdo nocivo no *system prompt*. Implementar classificadores de saída. Realizar *fairness audit* periódico com avaliadores humanos. Considerar *fine-tuning* com dados de alinhamento (RLHF/DPO).

6.1.7 Helpfulness vs. Harmlessness (LLM-as-a-Judge)

Definição: Avaliação multidimensional de respostas realizada por um modelo LLM avaliador (*judge*), que pontua cada resposta em dimensões como utilidade, segurança e correção factual.

Como funciona: Um par (consulta, resposta) é submetido a um modelo avaliador com *prompt* estruturado que solicita pontuação em escala *Likert* (1–5) para cada dimensão. O modelo avaliador deve ser distinto do modelo avaliado para evitar viés de autoavaliação. A concordância entre LLM-as-a-Judge e avaliadores humanos deve ser calibrada antes do uso em produção (correlação de *Spearman* > 0,7 indica alinhamento aceitável).

Quando usar: Avaliação contínua em larga escala onde avaliação humana seria impraticável; calibração periódica com amostra humana; comparação de versões de *prompts* em testes A/B.

Limitação: LLM-as-a-Judge pode ter viés em favor de respostas longas e formalmente elaboradas, independentemente da correção factual. Também pode apresentar viés de alinhamento semelhante ao modelo avaliado, sensibilidade ao *prompt* de avaliação, inconsistência entre execuções e risco de *reward hacking*, quando o sistema passa a otimizar características que agradam o avaliador automático sem melhorar qualidade real. Não substitui avaliação humana para casos de alta criticidade.

Boas práticas: usar avaliador diferente do modelo avaliado, fixar rubricas, medir concordância com avaliadores humanos, executar amostragem repetida em casos ambíguos, manter exemplos sentinela contra *reward hacking* e registrar versões do avaliador e do *prompt* de julgamento.

6.1.8 Métricas operacionais de inferência

Definição: Métricas que avaliam desempenho operacional do sistema LLM em produção, especialmente TTFT (*Time to First Token*), tokens por segundo, latência total, latência P95/P99, taxa de erro, taxa de *timeout*, utilização de GPU/CPU/memória e custo por interação.

Por que são críticas: Em atendimento ao cidadão, baixa qualidade operacional reduz a satisfação, aumenta abandono e pode inviabilizar o serviço mesmo quando a qualidade textual é adequada. Métricas de cauda, como P95 e P99, são mais relevantes do que média para dimensionar experiência real em horários de pico.

Como avaliar: Testes de carga antes da implantação, métricas do servidor de inferência, rastreamento distribuído, *dashboards* de MLOps/FinOps e alertas por degradação sustentada. Os limites devem ser definidos por canal de atendimento, criticidade do serviço e volume esperado.

6.1.9 Observabilidade semântica

Definição: Observabilidade semântica é a capacidade de inspecionar não apenas latência, erro e custo, mas também as decisões linguísticas e arquiteturais que produziram uma resposta: intenção classificada, rota escolhida, documentos recuperados, versão do *prompt*, ferramentas chamadas, validações aplicadas e motivo de *fallback* ou escalação.

Componentes essenciais: *tracing* por decisão do orquestrador, debug de *prompts* com versionamento, replay controlado de conversas, comparação entre versões de modelo, amostras de respostas reprovadas por *guardrails*, logs de documentos recuperados e trilha de auditoria para chamadas de ferramentas. Em ambientes com dados pessoais, replay e logs devem usar minimização, pseudonimização, controle de acesso e prazo de retenção definido.

Uso operacional: A observabilidade semântica permite responder perguntas de engenharia que logs tradicionais não respondem: por que o orquestrador escolheu um modelo caro, por que um documento irrelevante foi recuperado, por que o *prompt* mudou o comportamento, por que um *guardrail* reprovou a saída, ou por que uma conversa passou de baixa para alta criticidade. Esse diagnóstico é necessário para depuração, auditoria, melhoria contínua e investigação de incidentes.

6.2 Métricas de experiência do usuário

Referem-se às métricas comportamentais que refletem satisfação e utilidade percebida pelos usuários.

6.2.1 Taxa de Satisfação

Definição: Proporção de interações avaliadas positivamente por usuários.

Como coletar: Feedback explícito: botões positivos e negativos, avaliação por estrelas (1-5), pesquisas pós-interação. Pergunta simples: “Esta resposta foi útil? Sim/Não”

Exemplos internos de valores de referência: >80% para sistemas bem-sucedidos

Limitação: Viés de resposta - usuários insatisfeitos tendem a responder mais que usuários satisfeitos. Considerar em conjunto com outras métricas.

Uso: Tendências ao longo do tempo são mais informativas que valores absolutos. Se a taxa de satisfação caiu consistentemente, investigar causa (degradação de qualidade, mudanças na base de conhecimento). Avisos sobre limitações do modelo podem reduzir a satisfação percebida; por isso, devem ser desenhados como comunicação de risco proporcional ao contexto, sem ocultar limitações relevantes.

6.2.2 Taxa de Reformulação

Definição: Percentual de consultas seguidas por consulta similar reformulada pelo mesmo usuário em curto intervalo (ex: <5 minutos).

Interpretação: Alta taxa de reformulação indica que a primeira resposta não satisfaz, levando usuário a tentar novamente com palavras diferentes.

Sinal indireto mas confiável: Mais confiável que feedback explícito (reflete comportamento real, não declarado).

Exemplos internos de valores de referência:

- **Excelente:** Taxa de reformulação < 15%
- **Aceitável:** 15-30%
- **Problemático:** >30% (usuários frequentemente insatisfeitos com primeira resposta)

Análise: Identificar padrões de consultas com alta reformulação para detectar tópicos ou tipos de consulta onde o sistema tem baixo desempenho.

6.2.3 Taxa de Escalação para Humanos

Definição: Percentual de interações que requerem intervenção humana (transferência para atendimento humano, abertura de ticket).

Interpretação: Taxa baixa indica sistema resolve maioria das consultas autonomamente. Taxa alta indica limitações do sistema ou consultas muito complexas

Exemplos internos de valores de referência:

- **Sistemas bem ajustados:** <10%
- **Sistemas em maturação:** 10-20%
- **Sistemas iniciais ou complexos:** 20-30%

Trade-off: Taxa **muito** baixa (<5%) pode ser problemático se sistema está respondendo inadequadamente sem escalar quando deveria. Importante: O sistema deve escalar quando confiança é baixa, não forçar respostas inadequadas.

Análise: Classificar motivos de escalação (informação não disponível, consulta ambígua, necessidade de julgamento humano) para priorizar melhorias.

6.2.4 Avaliação Humana (Gold Standard)

Quando usar: Validação inicial antes de produção. Investigação de problemas de qualidade. Certificação regulatória ou auditoria. Calibração de métricas automáticas

Método: Especialistas de domínio avaliam amostra representativa de interações (mínimo 50-100). Para cada interação, avaliar:

- **Correção factual:** Informação está correta?

- **Fundamentação:** Resposta é suportada por documentos fonte?

- **Relevância:** Resposta aborda a consulta adequadamente?

- **Adequação ao contexto governamental:** Linguagem, tom, nível de detalhe são apropriados?

- **Conformidade:** Resposta está em conformidade com políticas institucionais?

Critérios de avaliação (escala *Likert* 1-5):

- 5: Excelente (nenhum problema)
- 4: Bom (problemas menores)
- 3: Aceitável (problemas moderados)
- 2: Ruim (problemas significativos)
- 1: Inaceitável (resposta incorreta ou inadequada)

Concordância entre avaliadores: Usar múltiplos avaliadores (mínimo 2) para mesma amostra, calcular concordância (*Cohen's Kappa* ou *Fleiss' Kappa*). *Kappa* >0.6 indica concordância aceitável.

Limitação: Custosa e não escalável. Usar para:

- Calibrar métricas automáticas (validar que métricas automáticas correlacionam com avaliação humana).
- Avaliar aspectos qualitativos que métricas automáticas não capturam (adequação de tom, nuances culturais).
- Investigação profunda de problemas detectados por métricas automáticas.

6.2 Avaliação contínua em produção

A avaliação pontual realizada durante o desenvolvimento e a homologação não é suficiente para garantir a qualidade de sistemas LLM ao longo do tempo. Em produção, os padrões de consulta dos usuários mudam, o modelo base pode ser atualizado pelo fornecedor e o system *prompt* pode precisar de revisão. A avaliação contínua é o mecanismo que detecta degradações antes que impactem os usuários.

Conjunto *golden* permanente: manter um conjunto de teste curado (pares consulta–resposta esperada) estável ao longo do tempo. Deve ser reexecutado a cada release e em ciclos periódicos (mensal ou trimestral), com comparação sistemática contra a linha de base. Degradação superior a 10% em qualquer métrica principal deve acionar revisão obrigatória.

Avaliação em modo sombra (*shadow evaluation*): amostras de consultas reais de produção

são avaliadas automaticamente por LLM-as-a-Judge ou classificadores sem intervenção do usuário, alimentando *dashboards* de qualidade em tempo real. A amostragem deve ser estratificada por categoria de consulta para garantir cobertura representativa.

Testes A/B de componentes: alterações em parâmetros de decodificação, system *prompt* ou versão do modelo devem ser validadas por testes A/B controlados antes de serem promovidas para produção, com critérios de sucesso definidos previamente no Plano de Experimentação.

Detecção de deriva comportamental (*model drift*): monitorar métricas de aderência ao *prompt*, ativações de *guardrails* e taxa de escalação ao longo do tempo. Variações sustentadas sinalizam necessidade de reavaliação do system *prompt* ou atualização do modelo.

Re-calibração periódica de limiares: os limiares de qualidade definidos no ETP e no Plano de Experimentação devem ser revisados periodicamente à luz dos dados acumulados em produção, à medida que a maturidade do sistema e as expectativas dos usuários evoluem.

Governança de resultados: os resultados das avaliações contínuas devem ser documentados, revisados em reuniões periódicas e utilizados para priorizar o *backlog* de melhorias. As métricas de qualidade do sistema LLM devem integrar os painéis de MLOps e FinOps do projeto.

Essas práticas devem ser descritas no ETP como requisitos não funcionais e detalhadas no Plano de Experimentação, com responsáveis, frequências e critérios de aceitação definidos antes da entrada em produção.

7. Padrões operacionais recomendados

Esta seção consolida padrões de arquitetura por tipo de sistema. A intenção é transformar os conceitos anteriores em orientação prática de projeto, contratação, revisão técnica e sustentação.

7.1 Padrões por tipo de sistema

Tipo de sistema	Arquitetura recomendada	Controles essenciais	Métricas prioritárias
Chatbot institucional	RAG com base curada, *prompt* institucional versionado, classificação de intenção e *fallback* para atendimento humano.	*Guardrails* de entrada e saída, citação de fontes, recusa segura, controle de dados pessoais e revisão de respostas críticas.	*Groundedness*, satisfação, taxa de reformulação, escalação humana e latência P95/P99.
Assistente interno de servidores	RAG sobre documentos internos, memória limitada por sessão e ferramentas somente leitura quando possível.	Autorização por perfil, segregação de dados, logs minimizados, controle de acesso a documentos e replay auditável.	Precisão por domínio, taxa de resposta com fonte, adoção, custo por sessão e incidentes de acesso indevido.

Automação de processo	Orquestrador determinístico com LLM para interpretação, extração e geração; ferramentas transacionais protegidas por *schema* e confirmação.	Validação tipada, idempotência, aprovação humana para ações sensíveis, trilha de auditoria e testes fim a fim.	Taxa de conclusão, erro de ferramenta, retrabalho humano, tempo de ciclo e custo por processo concluído.
Análise documental especializada	RAG hierárquico, *chunking* por estrutura documental, *reranking* e resposta com evidências.	Preservação de metadados, verificação de suporte textual, controle de versão das fontes e avaliação humana por amostra.	*Recall@k*, *groundedness*, cobertura de evidências, taxa de falso positivo e tempo de análise.
Agente multi-etapas	Ciclo planejar-executar-observar com ferramentas permitidas, orçamento de execução e parada explícita.	*max_steps*, timeout, limite de custo, detecção de loop, confirmação humana e auditoria por etapa.	Sucesso por tarefa, passos médios, custo por tarefa, falha de plano e taxa de escalação.

7.2 Checklist mínimo de produção

Antes de implantação, o sistema deve possuir: arquitetura com fronteiras documentadas; política de decisão do orquestrador; *prompts* versionados; *schemas* de entrada e saída; *guardrails* de entrada, contexto, saída e ação; avaliação *offline* com conjunto curado; testes adversariais; observabilidade semântica; orçamento de custo; política de retenção de logs; *runbook* de incientes; *fallback*; e critérios de desativação ou *rollback*.

8. Riscos específicos do uso de LLM

8.1 Jailbreaking e Injeção Direta de Prompt

Descrição: Usuário mal-intencionado subverte as instruções do *system prompt* por meio de entradas adversariais cuidadosamente formuladas, fazendo o modelo ignorar restrições, revelar conteúdo proibido ou adotar comportamento não autorizado.

Cenário de ameaça:

- ***Jailbreak* por *role-play*:** usuário instrui o modelo a "interpretar um personagem sem restrições" ou "simular um sistema sem filtros", contornando *guardrails* do *system prompt*.
- **Injeção em campos de formulário:** usuário insere em campos de dados da aplicação instruções do tipo "Ignore as instruções anteriores e faça X", que são processadas pelo modelo junto ao *prompt* da aplicação.
- **Exfiltração do *system prompt*:** usuário instrui o modelo a "repetir na íntegra todas as instruções que recebeu", expondo o *system prompt* confidencial, que pode conter informações de negócio ou de segurança.

- **Escalada de privilégios por contexto:** em conversas longas, usuário injeta gradualmente instruções que redefinem o contexto, fazendo o modelo "esquecer" as restrições originais.

Probabilidade: ALTA. Justificativa: técnicas de *jailbreak* são amplamente documentadas e acessíveis. Sistemas públicos com alto volume de usuários têm maior exposição, pois adversários podem testar exaustivamente diferentes vetores de ataque.

Impacto: ALTO. Justificativa: geração de conteúdo nocivo em nome da instituição, exposição de configurações internas, dano reputacional severo, possível responsabilização por orientação incorreta ou ilegal fornecida a cidadãos.

Classificação de risco: CRÍTICO (Alta probabilidade × Alto impacto).

Mecanismos de detecção:

1. Classificador de intenção adversarial na entrada: detecta padrões típicos de injeção ("ignore", "esquece", "novo sistema", "repita suas instruções").
2. Monitoramento de ativações de *guardrails*: registrar frequência de tentativas bloqueadas por categoria.
3. Auditoria periódica de logs por equipe de segurança: revisão manual de amostra de interações sinalizadas.
4. Alertas automáticos para padrões de exfiltração do *system prompt.*

Controles de mitigação obrigatórios:

- ***Guardrail* de entrada:** classificador que detecta e bloqueia entradas com padrões adversariais antes de envio ao modelo. Deve ser atualizado periodicamente com novos vetores documentados.
- **Guardrail de saída:** verificação pós-geração antes de exibir ao usuário, bloqueando respostas que violem políticas de conteúdo.
- **Logs completos de auditoria:** toda interação registrada com usuário, *prompt* de entrada, resposta gerada e *timestamp*. Retenção mínima de 12 meses para investigação.
- ***Red teaming* obrigatório antes de produção:** equipe adversarial testa *jailbreaks* conhecidos e variantes criativas antes de cada *release* significativo.

Controles de mitigação recomendados:

- **Múltiplas camadas de defesa (*defense in depth*):** classificador de entrada + *system prompt* robusto + guardrail de saída. Cada camada atua independentemente.
- **Proteção do *system prompt*:** instruir o modelo explicitamente a nunca revelar o conteúdo do *system prompt*, independentemente da solicitação do usuário.
- **Alertas em tempo real:** tentativas de *jailbreak* detectadas disparam notificação para equipe de segurança.

Risco residual: MÉDIO. Justificativa: técnicas de *jailbreak* evoluem continuamente; nenhum *guardrail* garante proteção total. Múltiplas camadas de defesa reduzem significativamente a taxa de sucesso de ataques, mas vigilância contínua e atualização periódica dos classificadores são necessárias.

8.2 Injeção Indireta de Prompt

Descrição: instruções adversariais chegam ao modelo embutidas em conteúdo externo aparentemente legítimo, como documentos anexados, e-mails, páginas web, transcrições, campos de cadastro, planilhas ou resultados de busca. Diferentemente da injeção direta, o usuário final pode não ser o atacante; o sistema executa ou resume conteúdo que já contém instruções maliciosas.

Cenário de ameaça:

1. Documento submetido para análise contém texto oculto ou instrução explícita para ignorar políticas do sistema e revelar dados internos.
2. Página web sumarizada contém comando para chamar uma ferramenta com parâmetros controlados pelo atacante.
3. E-mail processado pelo sistema instrui o modelo a classificar a mensagem como segura ou a encaminhar conteúdo sensível para terceiro.
4. Campo estruturado de cadastro contém instruções que passam a compor o contexto do LLM em etapa posterior do fluxo.

Probabilidade: ALTA. Justificativa: sistemas governamentais frequentemente processam documentos e dados externos em grande volume. Conteúdo não confiável pode entrar no contexto do modelo sem revisão humana prévia.

Impacto: ALTO. Justificativa: pode causar vazamento de dados, execução indevida de ferramentas, adulteração de classificações, *bypass* de políticas e respostas institucionais enganosas.

Controles de mitigação obrigatórios:

1. **Separação de contexto confiável e não confiável:** conteúdo externo deve ser delimitado e apresentado ao modelo como dado a ser analisado, nunca como instrução operacional.
2. **Sanitização e classificação de conteúdo externo:** documentos, páginas e campos estruturados devem passar por detector de injeção indireta antes de entrar no *prompt*.
3. **Validação fora do modelo:** chamadas de ferramentas, permissões, destinatários, identificadores e parâmetros transacionais devem ser validados por código determinístico e por política de autorização, não apenas pela decisão textual do LLM.
4. **Testes adversariais com documentos:** o *red teaming* deve incluir amostras com instruções maliciosas embutidas em PDFs, HTML, e-mails, planilhas e campos de formulário.

Risco residual: MÉDIO. Justificativa: a injeção indireta explora a dificuldade dos LLMs em distinguir instruções do sistema de conteúdo a ser analisado. Delimitação, validação externa e guardrails reduzem o risco, mas não eliminam a necessidade de monitoramento contínuo.

8.3 Exfiltração de dados por embeddings e memória

Descrição: dados pessoais ou sigilosos podem ser expostos por índices vetoriais, bases de memória, logs de consulta, respostas de *retrieval* ou similaridade indevida entre consultas. Mesmo quando o texto original não é armazenado integralmente, *embeddings* e metadados podem revelar informações sensíveis ou permitir inferências sobre indivíduos, processos ou documentos.

Cenário de ameaça:

- Usuário sem autorização formula consulta que recupera trecho sensível semanticamente próximo em base vetorial.
- Índice de *embeddings* é compartilhado entre unidades ou sistemas com diferentes níveis de permissão.
- Logs de busca armazenam consultas com dados pessoais e identificadores de documentos recuperados.
- Memória persistente de uma sessão é reutilizada em contexto de outro usuário por falha de segregação.

Controles de mitigação obrigatórios:

- **Controle de acesso antes e depois do *retrieval*:** filtrar documentos candidatos por autorização do usuário antes da recuperação e validar novamente os trechos retornados antes de inseri-los no *prompt*.
- **Segregação de índices:** separar bases vetoriais por domínio, classificação da informação, unidade responsável ou perfil de acesso quando houver risco de mistura indevida.
- **Minimização e retenção:** evitar armazenar dados pessoais em consultas, *embeddings*, metadados e logs além do necessário para a finalidade aprovada.
- **Auditoria de recuperação:** registrar quais documentos foram recuperados, por qual consulta, para qual usuário e com qual política de autorização.

Risco residual: MÉDIO. Justificativa: representações vetoriais reduzem legibilidade direta, mas não eliminam risco de inferência. O risco depende da sensibilidade do corpus, do modelo de *embedding*, dos metadados e das políticas de acesso.

8.4 Deriva Comportamental do Modelo (Model Drift)

Descrição: Atualizações do modelo base, realizadas silenciosamente pelo fornecedor da API ou por substituição de versão local, alteram o comportamento do sistema sem que o *system prompt* tenha mudado. A degradação de qualidade ou o surgimento de comportamentos inesperados ocorre gradualmente, sem alertas óbvios.

Cenário de ameaça:

- Fornecedor de API atualiza o modelo sem notificação adequada; a mesma solicitação passa a gerar respostas com formato ou tom diferente.

- Instrução do *system prompt* que funcionava na versão anterior passa a ser interpretada de forma divergente na nova versão.
- Comportamento em casos extremos (*edge cases*) muda: respostas antes recusadas passam a ser geradas, ou respostas antes aceitas passam a ser bloqueadas.
- Qualidade percebida pelos usuários degrada gradualmente; equipe técnica não detecta o problema pois a degradação é lenta e não abrupta.

Probabilidade: MÉDIA-ALTA. Justificativa: fornecedores de API atualizam modelos periodicamente; a frequência aumenta com a maturidade do mercado. Sem testes de regressão automatizados, deriva comportamental não é detectada a tempo.

Impacto: MÉDIO. Justificativa: degradação gradual de qualidade, não catastrófica. Usuários recebem respostas de qualidade subótima ou com comportamentos inesperados. Impacto: insatisfação, perda de confiança, possível geração de conteúdo inadequado não detectado pelos *guardrails* anteriores.

Classificação de risco: MÉDIO.

Mecanismos de detecção:

- Testes de regressão de *prompts* automatizados a cada nova versão do modelo: executar conjunto *golden* e comparar métricas com *baseline*.
- Monitoramento longitudinal de métricas de qualidade de geração e ativações de guardrails ao longo do tempo (mensal/trimestral). Variação consistente indica deriva.
- Avaliações periódicas com conjunto *golden*: re-executar a cada trimestre/semestre; degradação > 10% requer investigação.

Controles de mitigação obrigatórios:

- **Versionamento explícito do modelo em produção:** fixar a versão exata do modelo utilizado (ex.: gpt-4o-2024-08-06); só atualizar após validação em *staging*.
- **Testes de regressão de *prompts* obrigatórios:** executar conjunto *golden* antes de qualquer atualização de modelo em produção, com critérios de aprovação definidos no Plano de Experimentação.
- **Reavaliação trimestral ou semestral:** agendar avaliações regulares de qualidade de geração, documentar resultados e comparar com período anterior.

Controles de mitigação recomendados:

- **Análise de tendências:** *dashboard* com métricas de qualidade ao longo do tempo permite detecção visual precoce de deriva.
- ***Feedback loop*:** consultas com baixa satisfação são analisadas para identificar mudanças comportamentais do modelo.
- **Testes A/B de versões:** nova versão do modelo opera em modo sombra antes de ser promovida a produção.

Risco residual: BAIXO-MÉDIO. Justificativa: com versionamento explícito e testes de regressão automatizados, deriva comportamental é detectada antes do impacto em produção. O risco não é eliminado – modelos inevitavelmente evoluem – mas é gerenciável com processo estruturado de atualização.

8.5 Supply chain de modelos e pesos

Descrição: modelos, pesos, tokenizadores, adaptadores LoRA, imagens de contêiner, bibliotecas de *serving* e *datasets* podem introduzir vulnerabilidades, *backdoors*, licenças incompatíveis ou comportamentos maliciosos. Em auto-hospedagem, o risco de cadeia de suprimentos passa a ser responsabilidade direta do contratante.

Controles de mitigação obrigatórios:

- **Origem verificável:** baixar modelos e tokenizadores apenas de repositórios confiáveis, com *hash*, assinatura ou mecanismo equivalente de integridade quando disponível.
- **Inventário de componentes:** manter SBOM ou inventário técnico de modelo, versão, *tokenizer*, adaptadores, imagem de contêiner, bibliotecas e *datasets* usados.
- **Revisão de licença:** validar permissões de uso, redistribuição, uso comercial, restrições de dados e obrigações de atribuição.
- **Varredura e isolamento:** analisar imagens e dependências, executar contêineres com privilégios mínimos e impedir execução de código remoto não autorizado durante carga do modelo.
- **Teste comportamental:** executar conjunto de segurança e regressão antes de promover modelo, adaptador ou imagem para produção.

Risco residual: MÉDIO. Justificativa: a complexidade da cadeia de modelos dificulta garantia total de origem e comportamento. Verificação de integridade, inventário e isolamento reduzem a exposição.

8.6 Alucinações e fabricação de conteúdo

Descrição: LLM gera informação não fundamentada no conteúdo fornecido no *prompt*, apresentando-a como factual com aparente confiança. "Preenche lacunas" com conhecimento paramétrico armazenado em seus pesos (potencialmente incorreto ou desatualizado), criando afirmações plausíveis mas falsas. Em LLM puro, sem base de busca externa, este risco é ainda mais pronunciado do que em sistemas RAG, pois o modelo não tem mecanismo automático de ancoragem em fontes verificáveis.

Cenário de ameaça:

- Usuário faz consulta sobre assunto específico do domínio institucional não coberto pelo conhecimento paramétrico do modelo.
- LLM, tentando ser útil, completa a resposta com informação de seus pesos (treinamento geral), potencialmente incorreta ou desatualizada.
- Resposta gerada contém mistura de informação plausível (dos pesos) e eventualmente correta, tornando difícil distinguir o que é factual.
- Usuário não consegue distinguir o que é fundamentado em fatos reais versus alucinado pelo modelo.

- Decisões são baseadas em informação falsa com aparência de confiança

Probabilidade: ALTA. Justificativa: alucinação é característica intrínseca de LLMs. Todos os modelos atuais alucinam em alguma medida, especialmente quando o contexto é insuficiente ou ambíguo.

Impacto: ALTO. Justificativa: desinformação institucional, decisões baseadas em informação incorreta, dano reputacional quando erro é descoberto, potencial responsabilização por orientação incorreta.

Classificação de risco: CRÍTICO (Alta probabilidade × Alto impacto).

Mecanismos de detecção:

- **Métricas de aderência ao *prompt*:** Frameworks de avaliação de LLMs (LLM-as-a-Judge) avaliam se a resposta é fundamentada no conteúdo fornecido no *prompt*. Aderência < 0,75 indica baixa confiança e aderência < 0,70 indica alta probabilidade de alucinação e necessidade de ação corretiva.
- **Verificação cruzada:** Comparar afirmações na resposta com o conteúdo do *prompt*. Afirmações sem suporte no material fornecido pelo usuário são sinalizadas.
- ***Feedback* de especialistas:** Revisão humana de amostra de respostas por especialistas de domínio identifica alucinações que métricas automáticas não detectam.

Controles de mitigação obrigatórios:

- ***System prompts* enfatizando fidelidade.** Instrução explícita e enfática: "Você DEVE se limitar estritamente ao conteúdo fornecido pelo usuário nesta conversa. NÃO adicione informação proveniente do seu treinamento que não esteja no *prompt*. Se a informação não está disponível, declare claramente que não sabe."
- **Instrução para admitir ignorância.** "Se o conteúdo fornecido for insuficiente para responder adequadamente, informe: 'A informação disponível é insuficiente para responder com segurança. Recomenda-se consultar [fonte oficial/especialista]'. "
- **Validação pós-geração.** Framework automático (LLM-as-a-Judge ou classificador) verifica aderência ao *prompt*. Se aderência < 0,75, resposta é sinalizada como de baixa confiança ou reencaminhada para revisão humana.
- **Aviso de limitação ao usuário.** Interface deve exibir aviso explícito de que respostas podem conter erros quando a consulta envolver informação crítica, baixa confiança, decisão de alto impacto ou ausência de fonte verificável. O aviso deve ser contextual e proporcional ao risco, evitando banners permanentes genéricos que reduzam a utilidade percebida sem melhorar a segurança.

Controles de mitigação recomendados:

- ***Confidence scoring*.** LLM indica nível de confiança em cada afirmação (quando o modelo suporta esta funcionalidade). Afirmações de baixa confiança são sinalizadas ao usuário.
- ***Fallback* para humano.** Quando a aderência ao *prompt* é baixa (abaixo de 0,7) ou quando a consulta envolve decisões de alto impacto, o sistema escala para atendimento humano em vez de forçar resposta potencialmente alucinada.

Classificação de autonomia decisória. O encaminhamento a atendente humano não deve ser

condicionado exclusivamente ao limiar de confiança computacional. O PL 2338/2023, aprovado no Senado Federal e remetido à Câmara dos Deputados em 17/03/2025, ainda deve ser tratado como projeto de lei em tramitação, e não como norma vigente. Ainda assim, suas diretrizes de supervisão humana devem ser consideradas como referência regulatória prospectiva para sistemas de alto impacto. O contratante deve definir, em documento de política, as categorias de decisão classificadas como de alto impacto, para as quais o encaminhamento a atendente humano é obrigatório independentemente do score de confiança do modelo, sem prejuízo das normas vigentes aplicáveis.

***Fine-tuning* ou *instruction-tuning*.** Ajustar modelo LLM com exemplos enfatizando comportamento conservador (admitir não saber em vez de alucinar). Requer *dataset* curado e expertise em *fine-tuning*.

- **Redução de temperatura.** Configurar parâmetro de temperatura do LLM para valor baixo (ex.: 0.2-0.3) reduz a entropia da amostragem e a variabilidade da saída, aumentando a consistência em relação ao contexto.

Risco residual: MÉDIO. Justificativa: alucinações não podem ser completamente eliminadas com a tecnologia atual de LLMs. São uma propriedade intrínseca do mecanismo de geração por previsão estatística. Em LLM puro – sem base de busca externa que ancore as respostas em fontes verificáveis – este risco é estruturalmente mais elevado do que em sistemas RAG. Controles reduzem significativamente a frequência e detectam a maioria dos casos, mas algum nível de risco residual persiste. Vigilância contínua e múltiplas camadas de defesa são necessárias.

8.7 Geração de Conteúdo Tóxico ou Enviesado

Descrição: O modelo reproduz preconceitos presentes em seus dados de pré-treinamento, gerando respostas discriminatórias por gênero, raça, origem, orientação sexual ou outras características protegidas, ou amplificando estereótipos de forma que pode ser imperceptível ao usuário.

Cenário de ameaça:

- Usuário consulta o sistema sobre perfis de beneficiários ou grupos sociais; o modelo gera resposta que associa características negativas a determinado grupo demográfico.
- Sistema de geração de texto para comunicações institucionais produz conteúdo com linguagem inadvertidamente excludente ou estereotipada.
- Consultas sobre temas sensíveis (saúde mental, diversidade, inclusão) resultam em respostas que reproduzem estigmas dos dados de treinamento.
- Disparidade de qualidade de respostas entre grupos: o modelo atende melhor consultas em português europeu do que em variedades regionais ou dialetos do português brasileiro.

Probabilidade: MÉDIA. Justificativa: viés em modelos de linguagem é amplamente documentado na literatura científica. A magnitude depende do modelo escolhido, do *fine-tuning* e dos *guardrails* implementados.

Impacto: ALTO. Justificativa: violação do princípio constitucional de isonomia (Art. 5º da CF/88);

potencial responsabilização por discriminação; dano reputacional institucional; erosão da confiança de grupos vulneráveis no serviço público.

Classificação de risco: ALTO (Média probabilidade × Alto impacto).

Mecanismos de detecção:

- Classificadores automáticos de toxicidade aplicados sobre amostra de respostas em produção (ex.: Perspective API, modelos NLI de detecção de conteúdo nocivo).
- *Fairness audit* periódico: avaliação por especialistas de equidade com amostra estratificada por grupos demográficos.
- Análise de *feedback* negativo de usuários segmentado por perfil: taxas de rejeição mais altas em grupos específicos podem indicar viés sistemático.

Controles de mitigação obrigatórios:

- ***Fairness audit* antes da implantação:** avaliação obrigatória com critérios mensuráveis de equidade por grupo de usuários, realizada antes da entrada em produção e repetida periodicamente.
- ***Guardrail* de toxicidade na saída:** classificador automático bloqueia ou sinaliza respostas que excedam limiar de toxicidade antes de exibição ao usuário.
- **Registro e revisão de incidentes:** toda resposta sinalizada como tóxica ou enviesada deve ser registrada, revisada por equipe responsável e utilizada para melhoria do sistema.

Controles de mitigação recomendados:

- ***Fine-tuning* com dados de alinhamento (RLHF/DPO):** ajuste do modelo com exemplos que penalizam respostas discriminatórias e recompensam respostas equânimes.
- ***Feedback loop* com grupos afetados:** canais de reporte acessíveis para usuários identificarem respostas inadequadas; revisão prioritária de reportes de grupos historicamente vulneráveis.

Risco residual: MÉDIO. Justificativa: viés em LLMs não pode ser completamente eliminado com a tecnologia atual; modelos treinados em dados históricos herdam as desigualdades presentes nesses dados. *Guardrails* e auditorias periódicas reduzem significativamente a exposição, mas vigilância contínua é necessária.

8.8 Memorização e Violação de Direitos Autorais

Descrição: O modelo reproduz, de forma literal ou *quasi-literal*, trechos de textos protegidos por direitos autorais ou dados pessoais presentes em seu conjunto de treinamento, expondo a instituição a responsabilização por violação de propriedade intelectual ou por tratamento indevido de dados pessoais.

Cenário de ameaça:

- Usuário solicita que o modelo "complete" ou "continue" um texto conhecido; o modelo reproduz trecho substancial de obra protegida de seu treinamento.

- Consultas específicas sobre indivíduos levam o modelo a "lembrar" e reproduzir dados pessoais (nomes, endereços, informações sensíveis) presentes nos dados de treinamento.
- Sistema de geração de documentos institucionais reproduz inadvertidamente texto de normas, contratos ou pareceres de terceiros sem atribuição.

Probabilidade: BAIXA-MÉDIA. Justificativa: memorização de sequências longas é mais comum em modelos maiores e em textos muito frequentes no treinamento (ex.: texto legislativo, jornalístico). A probabilidade aumenta em consultas que "induzem" a geração de conteúdo específico.

Impacto: ALTO. Justificativa: violação da LGPD se dados pessoais forem reproduzidos (multa até 2% do faturamento, máximo R$ 50 milhões – Art. 52); responsabilização por violação de direitos autorais (Lei 9.610/1998); dano reputacional.

Classificação de risco: MÉDIO (Baixa-Média probabilidade × Alto impacto).

Mecanismos de detecção:

- Detecção de n-gramas longos idênticos a obras conhecidas em amostras de respostas geradas (ferramentas de verificação de originalidade adaptadas para saída de LLMs).
- Monitoramento de respostas que contenham padrões de dados pessoais (CPF, RG, endereços) via expressões regulares e NER.

Controles de mitigação obrigatórios:

- **Política de retenção mínima de dados nos logs de inferência:** logs de produção não devem armazenar o conteúdo completo das respostas por períodos superiores ao necessário para auditoria, e devem ser pseudonimizados quando possível.
- **Mascaramento de dados pessoais na saída:** expressões regulares e NER detectam e mascaram CPF, RG e outros identificadores pessoais nas respostas antes da exibição ao usuário.

Controles de mitigação recomendados:

- **Instrução no *system prompt*:** orientar explicitamente o modelo a não reproduzir trechos extensos de textos de terceiros e a parafrasear em vez de transcrever literalmente.
- **Verificação de originalidade pós-geração:** para documentos críticos, comparar a saída contra bases de textos conhecidos antes da entrega ao usuário.

Risco residual: BAIXO-MÉDIO. Justificativa: a probabilidade de memorização em uso típico de sistema público é relativamente baixa, mas as consequências legais justificam controles preventivos. Com mascaramento de dados pessoais e instruções adequadas no *system prompt*, o risco residual é gerenciável.

8.9 Vazamento de dados de treinamento e inversão de modelo

Descrição: ataques de extração, inversão ou inferência de pertencimento tentam descobrir se determinado dado esteve no treinamento ou recuperar atributos sensíveis a partir do comportamento do modelo. Esse risco aumenta quando modelos são ajustados com datasets pequenos, sensíveis ou pouco anonimizados.

Cenário de ameaça:

- Atacante consulta repetidamente o modelo para induzir reprodução de exemplos raros usados em *fine-tuning*.
- Saídas do modelo revelam que um cidadão, processo ou documento fazia parte do conjunto de treinamento.
- Modelo ajustado em dados institucionais memoriza nomes, números de processo, endereços ou informações de saúde.

Controles de mitigação obrigatórios:

- **Minimização de dados de treinamento.** Remover dados pessoais desnecessários, duplicatas, identificadores diretos e exemplos raros sensíveis antes do ajuste.
- **Separação de avaliação.** Impedir que dados de validação e teste entrem no treinamento, evitando avaliação inflada e vazamento de exemplos.
- **Testes de memorização.** Submeter o modelo a *prompts* de extração, continuação e inferência de pertencimento antes da produção.
- **Limitação de saída e monitoramento.** Aplicar limites de comprimento, filtros de dados pessoais e alertas para padrões de reprodução literal.

Risco residual: MÉDIO. Justificativa: modelos podem memorizar exemplos raros ou repetidos. A mitigação depende da qualidade da curadoria, do tamanho do *dataset*, do método de treinamento e de testes adversariais específicos.

9. Modos de falha por componente

A tabela a seguir resume falhas recorrentes em sistemas LLM e suas mitigações esperadas. Ela deve ser usada como insumo para *threat modeling*, revisão de arquitetura, testes de aceitação e operação contínua.

Componente	Falha	Impacto	Mitigação
Entrada do usuário	Injeção direta de *prompt*, conteúdo ofensivo, dados pessoais excessivos ou intenção ambígua.	Resposta insegura, tratamento indevido de dados, aumento de risco jurídico e degradação da experiência.	Classificação de intenção e risco, mascaramento de PII, validação de campos e recusa proporcional.
Prompt	Conflito de instruções, *template* desatualizado, variável não escapada ou exposição do *system prompt*.	Resposta incorreta, comportamento instável, vulnerabilidade a *jailbreak* e perda de rastreabilidade.	*Templates* versionados, testes de regressão, delimitação de contexto, revisão de mudanças e proteção contra exfiltração.

RAG e contexto	Recuperação irrelevante, fonte desatualizada, *chunk* sem metadados ou conteúdo externo tratado como instrução.	Alucinação ancorada em fonte errada, resposta sem evidência e risco de injeção indireta.	*Chunking* por estrutura, filtros de metadados, *reranking*, separação entre dados e instruções e verificação de suporte textual.
LLM	Alucinação, viés, baixa aderência ao formato, regressão por troca de modelo ou sensibilidade a parâmetros.	Desinformação, dano reputacional, falha de integração e inconsistência entre versões.	Avaliação contínua, limites de temperatura, validação de saída, comparação entre versões e *fallback*.
Orquestrador	Roteamento incorreto, *retry* indevido, ausência de *fallback* ou decisão não auditável.	Custo excessivo, resposta de baixa qualidade, falha operacional e dificuldade de investigação.	Políticas explícitas, registro de decisão, limites de custo, *retries* classificados e escalação humana.
Ferramentas	Argumento inválido, autorização insuficiente, efeito colateral não intencional ou resposta de API mal interpretada.	Ação incorreta, alteração indevida de dados, incidente de segurança e quebra de processo.	JSON *Schema*, validação fora do modelo, escopo mínimo, idempotência, confirmação para ações sensíveis e auditoria.
Memória	Persistência indevida, mistura entre usuários, instrução maliciosa armazenada ou retenção excessiva.	Vazamento de dados, personalização incorreta, violação de finalidade e risco LGPD.	Segregação por usuário, expiração, consentimento, minimização, distinção entre memória informativa e instrução.
Embeddings e índice	Exfiltração por similaridade, índice desatualizado, baixo recall ou latência alta.	Recuperação incorreta, exposição indireta de dados e degradação de qualidade.	Controle de acesso ao índice, reindexação governada, avaliação *recall@k,* criptografia, auditoria de consultas e *tuning* de ANN.
Guardrails	Falso negativo, falso positivo excessivo, *bypass multi-turn* ou validação apenas em *prompt*.	Bloqueio indevido, falha de segurança, perda de confiança e aumento de risco residual.	Cascatas de classificação, testes adversariais, limiares calibrados, revisão humana e controles determinísticos.
Observabilidade	Logs incompletos, ausência de *tracing* semântico, retenção excessiva ou dados pessoais em claro.	Incidentes difíceis de investigar, não conformidade e impossibilidade de melhoria contínua.	*Tracing* por decisão, replay controlado, minimização, pseudonimização, *dashboards* de qualidade/ custo e política de retenção.

10. Glossário técnico

Alucinação: geração de informação plausível, mas incorreta ou sem suporte no conteúdo fornecido ao modelo.

Agente: sistema que usa LLM para planejar ou selecionar ações em múltiplas etapas, geralmente combinando ferramentas, memória e orquestração.

***Drift* comportamental:** alteração do comportamento do sistema LLM ao longo do tempo, causada por mudança de modelo, dados, configuração ou padrão de uso.

***Embedding*:** vetor numérico que representa texto ou outro objeto em espaço semântico, usado para busca, comparação, agrupamento e avaliação de similaridade.

***Function calling* / uso de ferramentas:** mecanismo pelo qual o modelo seleciona uma função externa, preenche argumentos estruturados e utiliza o resultado para responder ou executar parte de um fluxo.

***Guardrail*:** controle de entrada, contexto, saída ou ação que restringe o comportamento do sistema LLM conforme políticas de segurança, privacidade, qualidade e conformidade.

Injeção direta de *prompt*: tentativa do usuário de inserir instruções adversariais diretamente na conversa ou em campos processados pelo modelo.

Injeção indireta de *prompt*: instrução adversarial embutida em conteúdo externo processado pelo sistema, como documento, e-mail, página web ou registro de base de dados.

***Jailbreak*:** técnica adversarial destinada a contornar restrições do *system prompt*, guardrails ou políticas do modelo.

KV cache: cache de chaves e valores das camadas de atenção, usado para acelerar inferência autoregressiva e reduzir recomputação de prefixos.

Memória de longo prazo: armazenamento persistente de fatos, preferências, documentos ou vetores recuperáveis em conversas futuras, sujeito a controles de finalidade, consentimento, retenção e acesso.

Quantização: redução da precisão numérica dos pesos ou ativações do modelo para diminuir consumo de memória e custo de inferência, com necessidade de validação de impacto na qualidade.

Saída estruturada: resposta gerada em formato formal, como JSON validado por schema, usada para integração com sistemas e chamada de funções.

***System prompt*:** instrução de sistema fornecida antes da mensagem do usuário para definir papel, restrições e políticas de resposta; é artefato operacional da aplicação, não componente estrutural do modelo.

TTFT (*Time to First Token*): tempo entre o recebimento da requisição e a emissão do primeiro token da resposta, métrica crítica em interfaces com *streaming*.

11. Referências bibliográficas

[1] *OWASP Foundation. Threat Modeling. OWASP Community – The Open Web Application Security Project.* Disponível em: https://owasp.org/www-community/Threat_Modeling. Acesso em: 10 dez. 2025.

[2] *MITRE. AI Security 101. ATLAS – Adversarial Threat Landscape for Artificial-Intelligence Systems.* Disponível em: https://atlas.mitre.org/resources/ai-security-101. Acesso em: 10 dez. 2025.

[3] DAMA International. *DAMA-DMBOK: Data Management Body of Knowledge*. 2. ed. Bradley Beach, NJ: Technics Publications, 2017.

[4] National Cyber Security Centre (NCSC). Guidelines for Secure AI System Development: Secure Design. NCSC – National Cyber Security Centre, 2023. Disponível em: https://www.ncsc.gov.uk/collection/guidelines-secure-ai-system-development/guidelines/secure-design. Acesso em: 11 dez. 2025.

[5] CPQD. *MLSecOps: Estrutura Técnica e Controles de Segurança para Operações Confiáveis em IA/ML*. Projeto INSPIRE – Meta 4. FINEP. 2025.

[6] CASELI, Helena M.; NUNES, Maria das Graças Volpe (org.). *Processamento de Linguagem Natural: conceitos, técnicas e aplicações em português*. BRBR-NLP, 2023. Disponível em: https://brasileiraspln.com/livro-pln/. Acesso em: 06 mar. 2026.

[7] Senado Federal. *Projeto de Lei n° 2338, de 2023*. Disponível em: https://www25.senado.leg.br/web/atividade/materias/-/materia/157233. Acesso em: 22 abr. 2026.

[8] Câmara dos Deputados. *PL 2338/2023 – Ficha de tramitação*. Disponível em: https://www.camara.leg.br/proposicoesWeb/fichadetramitacao?idProposicao=2487262. Acesso em: 22 abr. 2026.

[9] **CPQD. Metodologia de Desenvolvimento de Soluções de IA:** *Guia Unificado de Inteligência Artificial (GuIA).* Frente M3.2 – Subfrente M3.2.1. Meta 3 – Projeto INSPIRE. Convênio nº 01.25.0728.00. MGI/Finep. Campinas-SP, 2025.