

Como Reduzir Custos de Sistemas de AI com LLMs sem Sacrificar Qualidade

Revisão baseada em evidências de seis alavancas, seus limites e uma arquitetura de referência governada

Aridio Silva · Pesquisador Independente
ORCID: 0009-0008-2411-6995

© Aridio Silva @aridiosilva · Outubro 2026

23

Como Reduzir Custos de Sistemas de AI com LLMs sem Sacrificar Qualidade

Relatório técnico · Versão 1.1 · Outubro 2026

Aridio Silva · Independent Researcher · ORCID 0009-0008-2411-6995

<https://doi.org/10.5281/zenodo.23732760>

Como Reduzir Custos de Sistemas de AI com LLMs sem Sacrificar Qualidade

Uma revisão baseada em evidências de seis alavancas, seus limites e uma arquitetura de referência governada

Aridio Silva Pesquisador Independente, Brasil Criador do SGAEIA — Secure Governed Autonomous Edge Intelligence Architecture **ORCID:** [0009-0008-2411-6995](https://orcid.org/0009-0008-2411-6995)

© Aridio Silva @aridiosilva · Outubro 2026 · Versão 1.1 · Texto e figuras sob licença CC BY 4.0

Resumo

Em 26 de outubro de 2026, R. Bhardwaj publicou um infográfico afirmando que uma stack de GenAI e IA agêntica bem projetada pode cortar o gasto mensal com LLMs em 70%, de US\$ 10.000 para US\$ 2.926, mantendo a qualidade das respostas igual ou melhor [9]. Este relatório decompõe a proposta em suas partes: uma tese central, seis alavancas de custo, uma arquitetura de referência, uma decomposição de custo e uma tabela de qualidade antes e depois. Em seguida, confronta cada parte com fontes primárias: artigos revisados por pares, preprints do arXiv e a documentação atual das APIs dos três maiores provedores de modelos. A literatura sustenta todas as alavancas, muitas vezes com efeitos maiores do que o post assume. Cascatas e roteadores igualaram a acurácia do modelo mais forte por uma fração do custo em [10, 13, 39, 51]. Os provedores cobram 50% menos por trabalho em lote [4, 18, 40] e apenas 0,1× o preço de entrada nas leituras de prompt em cache [5]. Três fragilidades se destacam. O número de 70% é ilustrativo, não medido. Ele omite o custo extra que as próprias alavancas trazem (embeddings, roteadores, juízes, avaliação). E a tabela de qualidade reporta diferenças menores que o erro amostral de um conjunto de avaliação típico [35]. O relatório acrescenta um modelo formal de custo, condições de equilíbrio para cascatas e caches, uma análise de sensibilidade que produz um cenário ilustrativo de economia líquida de 61–68% sob premissas declaradas, sem o apresentar como estimativa empírica, uma árvore de KPIs centrada no custo por tarefa bem-sucedida [14], um roteiro de implementação em fases, um glossário e uma bibliografia auditada de 56 fontes com links verificados em 27 de outubro de 2026.

Palavras-chave: LLMOps; custo de inferência; roteamento de modelos; cascatas de LLM; geração aumentada por recuperação; prompt caching; cache semântico; portões de qualidade; cost-of-pass; governança de IA.

Sumário

1. [Introdução](#)
2. [Método](#)
3. [A tese central](#)
4. [Um modelo de custo para sistemas com LLM](#)
5. [As seis alavancas](#)
6. [Arquitetura de referência](#)
7. [Motor de otimização de contexto](#)
8. [Cache: de prompt, semântico e KV](#)
9. [Roteamento de modelos e cascatas](#)
10. [Portões de qualidade e escalonamento](#)
11. [Decomposição do custo e sensibilidade](#)
12. [Avaliação e KPIs](#)

13. [Observabilidade e governança](#)
14. [Relação com o SGAEIA](#)
15. [Roteiro de implementação](#)
16. [Discussão e ameaças à validade](#)
17. [Conclusão](#)
18. [Glossário](#)
19. [Referências \(auditadas\)](#)

1 Introdução

1.1 Contexto

A inferência de LLMs virou custo operacional recorrente, não despesa de pesquisa. Cada requisição é cobrada por token, e os preços por token entre modelos diferiam em até duas ordens de grandeza quando o FrugalGPT os levantou [10]. Sistemas agênticos agravam o quadro. Uma única requisição de usuário pode disparar muitas chamadas de modelo, chamadas de ferramentas e novas tentativas [2, 50]. As equipes enfrentam, portanto, um problema de engenharia: entregar a qualidade de que a tarefa precisa pelo menor custo por resultado bem-sucedido.

1.2 Objeto de análise

O objeto é um post e infográfico de Rajeev Bhardwaj no LinkedIn [9]. O texto diz: cortar custos com LLMs "em até 70%", sem comprometer a inteligência, por meio de roteamento de modelos, contexto, cache e fluxos agênticos mais inteligentes. O infográfico tem seis partes:

- um resultado de destaque (de US\$ 10.000 para US\$ 2.926 por mês);
- seis "alavancas-chave";
- uma arquitetura de referência com seis etapas e uma camada de governança;
- uma cascata de "decomposição ilustrativa do custo";
- uma tabela de qualidade comparando métricas de base e otimizadas;
- uma mensagem central: *gaste mais inteligência só quando a tarefa realmente precisar*.

1.3 Contribuições

1. Uma representação estruturada da proposta, redesenhada em 13 figuras originais em inglês e português.
2. Para cada componente, a literatura primária que o sustenta, qualifica ou contradiz, com os conceitos explicados para um leitor técnico não especialista.
3. Um modelo formal de custo (Eq. 1) que liga cada alavanca a um fator do gasto, mais as condições de equilíbrio para cascatas (Eq. 4) e para prompt caching (Eq. 3).
4. Uma conferência da aritmética do post e uma análise de sensibilidade da afirmação de 70% que inclui o custo extra trazido pelas alavancas.
5. Uma árvore de KPIs, um roteiro em fases com portões de saída, um glossário e uma bibliografia auditada.

2 Método

A revisão seguiu quatro passos.

1. **Decomposição da fonte.** Cada afirmação do infográfico foi transcrita e agrupada por componente: tese, alavancas, arquitetura, decomposição, tabela de qualidade, mensagem central. O texto e a data de publicação do post foram lidos da página pública do LinkedIn.
2. **Seleção da literatura.** Para cada componente buscamos (a) artigos revisados por pares ou preprints do arXiv bem citados que definem ou medem a técnica, (b) a documentação dos próprios provedores para preços e limites e (c)

normas reconhecidas de governança e confiabilidade. Fontes primárias tiveram preferência sobre resumos em blogs.

3. **Auditoria.** Todas as referências foram resolvidas em 27 de outubro de 2026. Os registros do arXiv foram conferidos na API do arXiv quanto a título, autores e data de submissão. Cada uma das demais URLs foi acessada e seu status HTTP registrado. Os números citados neste relatório (por exemplo, "até 98%" para o FrugalGPT, ou o multiplicador de 0,1× na leitura de cache) foram copiados do resumo da fonte ou da página de documentação, não de resumos secundários.
4. **Síntese e ampliação.** As afirmações foram reconstruídas em um modelo de custo e testadas com cálculos analíticos e de sensibilidade simples. Onde o post não traz dados, as figuras são marcadas como *ilustrativas*.

Técnicas de eficiência do lado do modelo, como quantização, decodificação especulativa e escalonamento em sistemas de serving, são revisadas em outros trabalhos [33, 54] e ficam fora do escopo deste relatório, que trata o modelo como uma API com preço.

Limitações do método. Esta é uma revisão estruturada, não uma revisão sistemática com protocolo registrado. Nenhum experimento foi executado. Os preços dos provedores mudam com frequência e são citados como lidos na data da auditoria.

3 A tese central

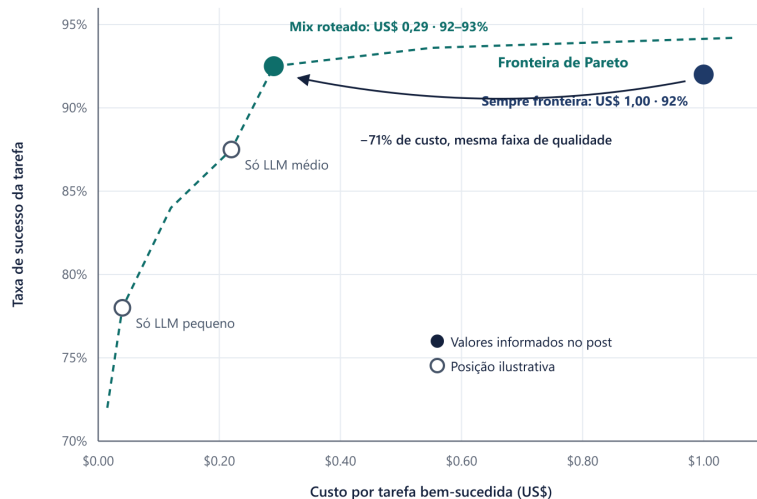
3.1 Enunciado

A tese é que custo e qualidade não precisam ser trocados um pelo outro de forma geral. A maioria das requisições de uma carga real é fácil. Pagar preço de modelo de fronteira por elas compra capacidade que não é usada. Um sistema que ajusta a capacidade à dificuldade pode, portanto, ocupar um ponto melhor da curva custo–qualidade do que outro que usa sempre o maior modelo.

FIGURA 1

Tese central: comprar qualidade onde ela é necessária, não em toda parte

Plano custo–qualidade (conceitual)



Fronteira de Pareto

Configurações em que nada é mais barato sem também ser pior.

Ponto dominado

Enviar toda requisição ao modelo de fronteira paga por capacidade que a maioria não usa.

Ponto de operação

Fixe primeiro um piso de qualidade (SLO) e então minimize o custo por tarefa bem-sucedida.

Mensagem central

Gaste mais inteligência só quando a tarefa realmente precisar.

Baseado em R. Bhardwaj (2026), post no LinkedIn, redesenhado e ampliado. Conceito de fronteira: Chen et al. 2023; Kapoor et al. 2024; Erol et al. 2025.
© Aridio Silva @aridiosilva · Outubro 2026 · Pesquisador Independente · ORCID 0009-0008-2411-6995 · Figuras sob licença CC BY 4.0

Figura 1. A tese central no plano custo–qualidade. Os dois pontos cheios são os valores de base e otimizado informados no post; os pontos vazados e a fronteira são ilustrativos. © 2026 Aridio Silva (@aridiosilva), Pesquisador Independente, ORCID 0009-0008-2411-6995. CC BY 4.0.

3.2 A visão de Pareto

Uma configuração é *Pareto-ótima* quando nenhuma outra é mais barata sem também ser pior. Kapoor et al. mostraram que benchmarks de agentes que reportam só acurácia empurram a área para agentes "desnecessariamente complexos e caros", e defendem otimizar custo e acurácia em conjunto [25]. O HELM já defendia avaliação multimétrica que incluía eficiência [28]. Nessa visão, o baseline "sempre fronteira" do post é um ponto *dominado* (Figura 1). O mix roteado alcança a mesma faixa de sucesso com 29% do custo por tarefa bem-sucedida.

3.3 A unidade certa: custo por tarefa bem-sucedida

A escolha mais importante do infográfico é a última linha da tabela de qualidade: **custo por tarefa bem-sucedida** (de US\$ 1,00 para US\$ 0,29). Erol et al. formalizam a mesma ideia como *cost-of-pass*: o gasto esperado para obter uma solução correta [14]. Com custo por tentativa c e probabilidade de sucesso p :

$$\text{CPTS} = c / p$$

Essa métrica impede que um modelo barato mas pouco confiável pareça bom. Um modelo que custa um décimo mas acerta metade das vezes é só cinco vezes mais barato por sucesso, e isso antes do custo de tratar suas falhas.

Pontos fortes da tese

- Coincide com o achado empírico de que roteadores e cascatas mantêm a qualidade por uma fração do custo [10, 13, 39].
- Oferece uma única métrica-alvo legível pelo negócio.
- Vale para agentes tanto quanto para chat [25].

Limites

- Depende da carga. Se a maioria das requisições for difícil, há pouco a desviar.
- "Mesma qualidade" precisa ser demonstrada estatisticamente, não afirmada (Seção 12).
- A qualidade na cauda longa de consultas raras e críticas pode cair enquanto a média parece boa.

4 Um modelo de custo para sistemas com LLM

O post lista alavancas mas não diz sobre qual parte da conta cada uma age. Tornamos isso explícito com um modelo de custo mensal:

$$C = N \cdot (1 - h) \cdot k \cdot [p_{in}(m) \cdot (T_u + \delta \cdot T_c) + p_{out}(m) \cdot T_{out}] \cdot (1 - \beta \cdot b) + C_{ovh}(1)$$

Aqui N é o número de requisições por mês, h a taxa de acerto do cache semântico, k o número médio de chamadas de LLM por requisição, m o modelo escolhido com seus preços de entrada e saída p , T_u e T_c os tokens de entrada sem e com cache, δ o multiplicador de preço da leitura em cache, T_{out} os tokens de saída, β o desconto de lote, b a parcela do tráfego enviada em lote e C_{ovh} o custo extra que a própria otimização acrescenta.

FIGURA 2

Modelo de custo: cada alavanca ataca um fator diferente

Gasto mensal com LLM decomposto

$$C = N \cdot (1 - h) \cdot k \cdot [p_{in}(m) \cdot (T_u + \delta \cdot T_c) + p_{out}(m) \cdot T_{out}] \cdot (1 - \beta \cdot b) + C_{ovh}$$

N

Requisições por mês

A demanda é dada. As alavancas atuam no custo de cada requisição.

Não é alavanca

1 - h

Taxa de falha do cache semântico

Um acerto pula recuperação, geração e checagens após uma busca.

A4 Cache semântico

k

Chamadas de LLM por requisição

Passos de agente, loops de ferramentas, novas tentativas e chamadas duplicadas.

A5 Loop agêntico

m → p(m)

Modelo escolhido → preço do token

Os preços de API diferiam em até duas ordens de grandeza (Chen et al. 2023).

A1 Roteamento

T_u

Tokens de entrada sem cache

Trechos recuperados, histórico e instruções cobrados a preço cheio.

A2 Contexto

δ · T_c

Tokens de prefixo em cache

Anthropic: leitura δ = 0,1×, escrita de 5 min 1,25×, OpenAI: até 95% de desconto.

A3 Prompt cache

T_{out}

Tokens de saída

Costumam custar várias vezes a entrada. Limite com `formatos` e `max_tokens`.

A2 Contexto

1 - β · b

Desconto de lote na parcela em lote

β = 0,5 na Anthropic, OpenAI e Google. b = parcela que pode esperar até 24 h.

A6 Lote

C_{ovh}

Custo extra que as alavancas trazem

Embeddings, banco vetorial, roteador, juizes e rodadas de avaliação.

Custo novo

Modelo de custo do autor. Multiplicadores de preço: documentação das APIs Anthropic, OpenAI e Google (acesso em 2026-10-27).

© Aridio Silva @aridiosilva · Outubro 2026 · Pesquisador Independente · ORCID 0009-0008-2411-6995 · Figuras sob licença CC BY 4.0

Figura 2. Cada alavanca ataca um fator diferente da Eq. 1. O cartão vermelho é o custo extra que as próprias alavancas introduzem e que o post omite. © 2026 Aridio Silva (@aridiosilva), Pesquisador Independente, ORCID 0009-0008-2411-6995. CC BY 4.0.

Duas propriedades da Eq. 1 explicam o formato da cascata do post. Primeiro, os fatores se **multiplicam**, então as economias se compõem: cinco reduções de 15%, 40%, 25%, 15% e 10% dão $1 - 0,85 \cdot 0,60 \cdot 0,75 \cdot 0,85 \cdot 0,90 = 70,7\%$, e não os 105% que uma soma ingênua sugeriria. Segundo, alavancas sobre o mesmo fator **não se acumulam por inteiro**. Depois que o roteamento levou o tráfego para um modelo mais barato, cortar seus tokens economiza dinheiro ao preço mais barato. A ordem em que as economias são contadas muda quanto cada etapa parece contribuir, mas não o produto final.

O processamento em lote mostra como ler o modelo. Os três grandes provedores dão 50% de desconto ao trabalho em lote [4, 18, 40], então $\beta = 0,5$. A etapa de 10% do post implica, portanto, $b = 20\%$: uma requisição em cada cinco pode esperar até 24 horas.

5 As seis alavancas

FIGURA 3

As seis alavancas: alvo, evidência, risco e trade-off

Síntese da revisão da literatura

Alavanca	Fator	Efeito relatado (fonte)	Risco	Esforço	Principal trade-off
1 Roteamento inteligente de modelos	$p(m)$	Corte de até 98% com acurácia do GPT-4 (FrugalGPT); >2× mais barato (RouteLLM); 40% menos chamadas ao modelo grande (Hybrid LLM).	med-alto	alto	Consultas difíceis mal roteadas; roteador exige dados e manutenção.
2 Engenharia de contexto	T_u, T_{out}	Compressão de até 20× com pouca perda (LLMLingua); evidência no meio do contexto é a pior usada (Liu 2023).	medio	medio	Cortar demais perde recall; contexto demais adiciona ruído.
3 Prompt caching	$\delta \cdot T_c$	Leitura a 0,1× do preço de entrada (Anthropic); até 95% de desconto (OpenAI); TTFT 8× menor em GPU (Prompt Cache).	baixo	baixo	Exige prefixo estável; custo de escrita; cache compartilhado vaza tempo.
4 Cache semântico	h	~31% das consultas eram repetidas no estudo do MeanCache; GPTCache open source.	alto	medio	Acertos falsos, respostas desatualizadas e vazamento entre tenants.
5 Otimização do loop agêntico	k	Baselines simples igualaram agentes SOTA caros por uma fração do custo (Kapoor 2024).	baixo-med	medio	Menos passos e tentativas vs. robustez em tarefas difíceis.
6 Processamento em lote	$\beta \cdot b$	50% de desconto, resultado em até 24 h (Anthropic, OpenAI, Google); acumula com cache (Anthropic).	baixo	baixo	Só para trabalho que pode esperar; adiciona fila.

Fontes: Chen 2023; Ong 2024; Ding 2024; Jiang 2023; Gill 2024; Kapoor 2024; docs Anthropic, OpenAI e Google (2026).
 © Aridio Silva @aridiosilva · Outubro 2026 · Pesquisador Independente · ORCID 0009-0008-2411-6995 · Figuras sob licença CC BY 4.0

Figura 3. As seis alavancas, o fator da Eq. 1 que cada uma ataca, o efeito mais forte relatado na literatura e o principal trade-off. © 2026 Aridio Silva (@aridiosilva), Pesquisador Independente, ORCID 0009-0008-2411-6995. CC BY 4.0.

5.1 Alavanca 1: roteamento inteligente de modelos

Conceito. Um roteador envia cada requisição ao modelo mais barato do qual se espera uma boa resposta. A cascata é uma variante sequencial: tenta primeiro um modelo barato, verifica a resposta e escala só se a verificação falhar.

Evidências.

- O FrugalGPT aprendeu qual combinação de APIs de LLM consultar por requisição. Igualou o desempenho do GPT-4 "com até 98% de redução de custo", ou melhorou a acurácia em 4% pelo mesmo custo [10].
- O Hybrid LLM roteou entre um modelo pequeno e um grande pela dificuldade prevista e fez "até 40% menos chamadas ao modelo grande, sem queda na qualidade da resposta" [13].
- O RouteLLM treinou roteadores com dados de preferência humana e reduziu o custo "em mais de 2 vezes em certos casos". Seus roteadores continuaram funcionando quando os modelos forte e fraco foram trocados no teste [30, 39].
- Cascatas por consistência de resposta atingiram acurácia de raciocínio no nível do GPT-4 com cerca de 40% do seu custo [51].

- O AutoMix usou autoverificação few-shot e um roteador POMDP para cortar o custo em mais de 50% com desempenho comparável [1].

Trade-offs. Roteadores precisam de dados rotulados ou de preferência e se desatualizam quando os modelos mudam. Um veredito "fácil" errado sobre uma requisição difícil prejudica mais a qualidade justamente nas consultas raras e críticas. Cascatas pagam duas vezes por cada requisição escalada (Seção 9).

5.2 Alavanca 2: engenharia de contexto

Conceito. Engenharia de contexto é decidir quais tokens entram na janela de contexto do modelo e em que ordem. A Anthropic a descreve como encontrar o menor conjunto possível de tokens de alto sinal para a tarefa [3].

Evidências.

- Os modelos usam pior a informação colocada no meio de um contexto longo do que a informação no início ou no fim ("lost in the middle") [29].
- O desempenho cai à medida que a entrada cresce, mesmo em tarefas simples [11].
- O LLMingua comprimiu prompts "até 20x" com pouca perda [23, 34]. O LongLLMingua estendeu isso para RAG de contexto longo [24].
- Prompting com consciência de orçamento também pode encurtar a saída de raciocínio [22].

Menos tokens economizam dinheiro e podem também *eleva*r a qualidade.

Trade-offs. Cortar demais perde justamente o trecho que importava (perda de recall). A compressão acrescenta uma chamada de modelo e latência. Orçamentos agressivos exigem avaliação por tipo de tarefa.

5.3 Alavanca 3: prompt caching

Conceito. Quando muitas requisições compartilham o mesmo prefixo de prompt (instruções de sistema, schemas de ferramentas, exemplos few-shot, um documento longo), o provedor pode reutilizar o estado de atenção já processado desse prefixo em vez de recalculá-lo. Gim et al. mostraram a base de sistemas disso: reutilizar estados de atenção reduziu o tempo até o primeiro token em 8x na GPU e 60x na CPU, sem alterar as saídas [17].

Condições atuais dos provedores (auditadas em 27/10/2026).

- **Anthropic:** leitura em cache a 0,1x do preço base de entrada (menos em alguns modelos novos). Escrita a 1,25x com TTL de 5 minutos ou 2x com TTL de 1 hora. Esses multiplicadores se acumulam com o desconto de lote [5].
- **OpenAI:** o cache é ativado por padrão nos modelos suportados, com a entrada em cache "com desconto de até 95%" e prefixo mínimo cacheável de 1.024 tokens nos modelos recentes [41].
- **Google:** cache implícito no Gemini, com cache explícito disponível em alguns modelos [19].

Trade-offs. O prefixo precisa ser idêntico byte a byte entre requisições, então os templates devem pôr o conteúdo estável primeiro e o variável por último. A escrita custa mais que a entrada comum. Gu et al. constataram que caches compartilhados entre usuários em sete provedores de API criam um canal lateral de tempo que pode vaziar informação sobre prompts de outros usuários [20].

5.4 Alavanca 4: cache semântico

Conceito. Um cache semântico guarda respostas inteiras e devolve uma delas quando uma nova consulta tem *significado próximo* ao de uma anterior, medido por similaridade de embeddings [45]. O GPtCache é a implementação open source de referência [7, 55]. O MeanCache relata que cerca de 31% das consultas em seu cenário eram repetidas, e alerta que caches semânticos ingênuos têm taxas de acerto e falha falsos "inaceitáveis" [16].

Trade-offs. Um acerto falso devolve uma resposta confiante a outra pergunta. As respostas ficam desatualizadas quando as fontes mudam. Um cache compartilhado entre tenants pode vaziar dados. A Seção 8 traz os controles necessários.

5.5 Alavanca 5: otimização do loop agêntico

Conceito. Agentes intercalam raciocínio e uso de ferramentas [50]. Cada passo é uma chamada paga, e passos redundantes, novas tentativas e repasses multiplicam o k da Eq. 1.

Evidências. Kapoor et al. constataram que baselines simples igualaram agentes de ponta em acurácia com custo muito menor [25]. A orientação da Anthropic é começar pelo padrão mais simples que funcione (uma única chamada, depois um fluxo fixo) e acrescentar autonomia só quando ela se paga [2].

Trade-offs. Limites rígidos de passos e tentativas podem interromper um agente pouco antes de ele ter sucesso. Os limites devem ser definidos por tipo de tarefa a partir dos traces.

5.6 Alavanca 6: processamento em lote

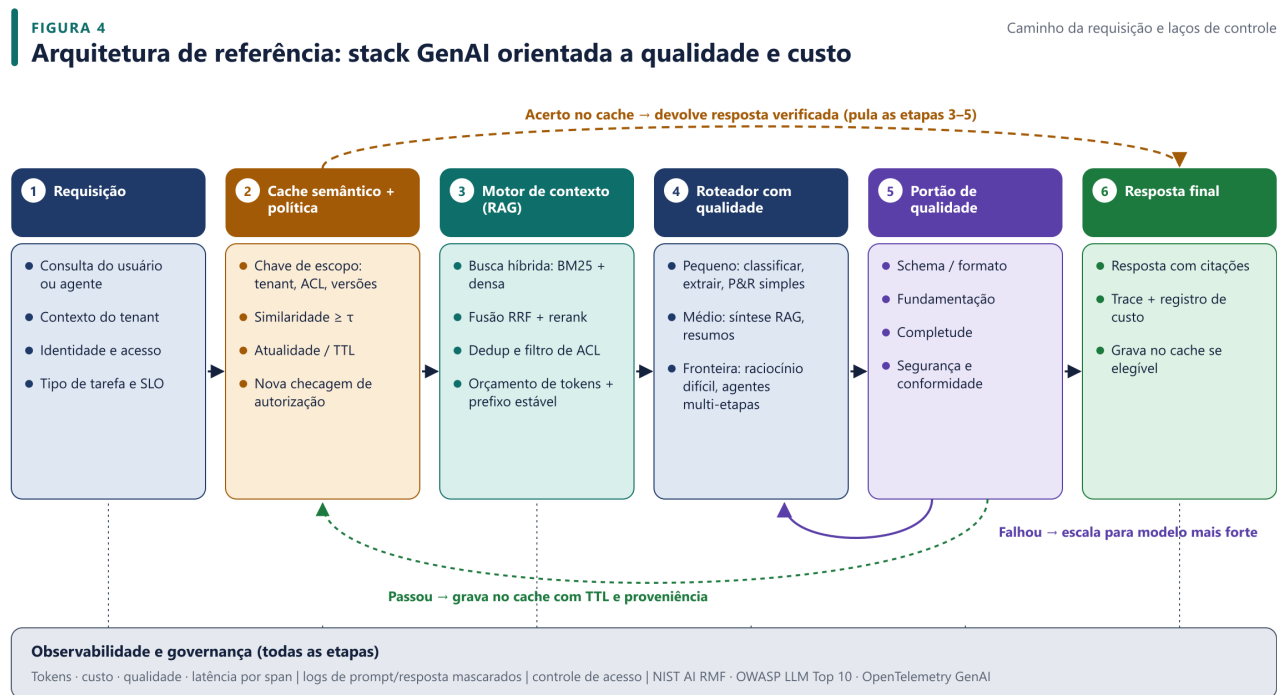
Conceito. O trabalho que não precisa de resposta imediata é enviado de forma assíncrona, em bloco. Exemplos são enriquecimento de dados, classificação de backlogs, rodadas de avaliação e geração de relatórios.

Evidências. Anthropic, OpenAI e Google cobram requisições em lote a 50% do preço síncrono e devolvem os resultados em até 24 horas, muitas vezes bem antes [4, 18, 40].

Trade-offs. Só se aplica a trabalho que tolera latência, e acrescenta código de fila e de tratamento de resultados.

6 Arquitetura de referência

O post organiza as alavancas em um pipeline. Nós o redesenhamos com os laços de controle explícitos (Figura 4).



Baseado em R. Bhardwaj (2026), post no LinkedIn, redesenhado e ampliado. Governança: NIST AI 100-1, NIST AI 600-1, OWASP LLM Top 10 (2025), convenções GenAI do OpenTelemetry.
© Aridio Silva @aridiosilva · Outubro 2026 · Pesquisador Independente · ORCID 0009-0008-2411-6995 · Figuras sob licença CC BY 4.0

Figura 4. Arquitetura de referência redesenhada a partir do post. Três laços de controle aparecem de forma explícita: desvio por acerto no cache (âmbar), escalonamento (violeta) e gravação no cache após o portão (verde). © 2026 Aridio Silva (@aridiosilva), Pesquisador Independente, ORCID 0009-0008-2411-6995. CC BY 4.0.

- 1. Requisição.** Leva a consulta, o tenant, a identidade e os direitos de acesso, e o tipo de tarefa. O tipo de tarefa é o sinal de roteamento mais barato e deve ser capturado na borda.
- 2. Cache semântico e checagem de política.** Devolve uma resposta salva só se ela for autorizada, atual e semanticamente equivalente (Seção 8).
- 3. Motor de otimização de contexto.** Busca híbrida, rerank, filtro e orçamento de tokens (Seção 7). É a geração aumentada por recuperação introduzida por Lewis et al. [27].
- 4. Roteador com consciência de qualidade.** Escolhe um modelo pequeno, médio ou de fronteira (Seção 9).
- 5. Portão de qualidade.** Verifica formato, fundamentação, completude e segurança, e escala em caso de falha (Seção 10).

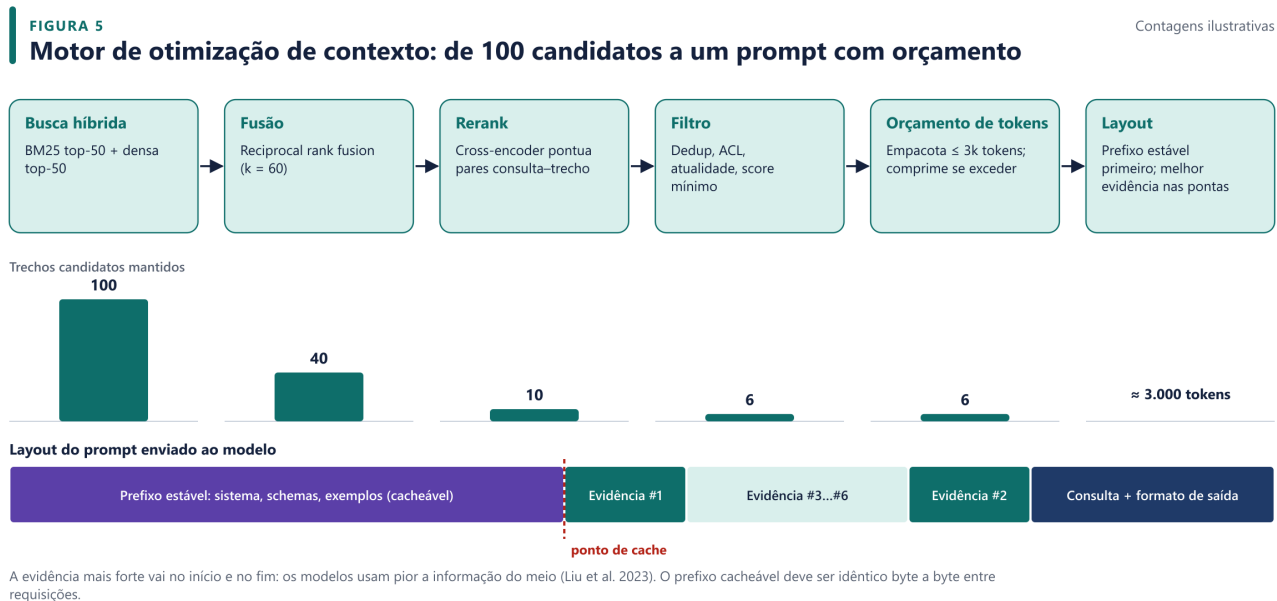
6. **Resposta final.** Devolve a resposta com citações e emite um trace com custo.

Uma camada transversal de **observabilidade e governança** registra tokens, custo, qualidade e latência por etapa (Seção 13).

Três princípios de projeto decorrem da literatura:

- **Decisões baratas primeiro.** A busca no cache e as checagens determinísticas custam muito menos que uma chamada de modelo, então rodam antes dela.
- **Fechar o laço.** Só respostas que passaram no portão são gravadas no cache, o que evita que erros sejam repetidos.
- **Preferir recuperação autocorretiva.** Self-RAG [6] e Corrective RAG [49] acrescentam checagens da qualidade da recuperação dentro do laço de geração. São extensões naturais das etapas 3 e 5.

7 Motor de otimização de contexto



Fontes: Robertson & Zaragoza 2009 (BM25); Cormack et al. 2009 (RRF); Nogueira & Cho 2019 (rerank); Liu et al. 2023; Jiang et al. 2023.
© Aridio Silva @aridiosilva · Outubro 2026 · Pesquisador Independente · ORCID 0009-0008-2411-6995 · Figuras sob licença CC BY 4.0

Figura 5. O motor de contexto reduz cerca de 100 trechos candidatos a um prompt com orçamento. Contagens ilustrativas. O layout põe o prefixo cacheável primeiro e a evidência mais forte nas pontas. © 2026 Aridio Silva (@aridiosilva), Pesquisador Independente, ORCID 0009-0008-2411-6995. CC BY 4.0.

7.1 Busca híbrida

A busca lexical com BM25 pontua documentos por frequência do termo, frequência inversa nos documentos e normalização por tamanho [46]. Ela lida bem com identificadores exatos, códigos e termos raros. A busca densa projeta consultas e trechos em um espaço vetorial [45] e encontra paráfrases. Índices de vizinhos mais próximos aproximados, como o HNSW, a tornam rápida em escala [31]. Rodar as duas e fundir os resultados cobre as fraquezas de cada uma.

7.2 Fusão com RRF

A reciprocal rank fusion funde listas ranqueadas sem calibrar seus scores [12]:

$$\text{RRF}(d) = \sum r \square R \ 1 / (k + \text{rankr}(d)), \ k = 60(2)$$

7.3 Rerank e filtragem

Um cross-encoder lê a consulta e cada trecho candidato em conjunto e pontua a relevância com muito mais precisão que o recuperador de primeira etapa, a um custo maior por par [38]. LLMs também podem atuar como rerankers [48]. Depois do rerank, filtros removem quase duplicatas, trechos que o usuário não pode ver (ACL), versões desatualizadas e trechos abaixo de um score mínimo.

7.4 Orçamento de tokens e layout

O empacotador preenche um orçamento fixo com os melhores trechos. Se o orçamento for excedido, ele comprime [24] em vez de truncar às cegas. O layout segue duas regras:

- O conteúdo estável (prompt de sistema, schemas de ferramentas, exemplos) vem primeiro para poder ser cacheado [5].
- A evidência mais forte vai no início e no fim da parte dinâmica, onde os modelos a usam melhor [29].

Tabela 1. Técnicas do motor de contexto: benefício e custo

Técnica	Principal benefício	Principal custo ou risco
Híbrida BM25 + densa	Recall em termos exatos e em paráfrases	Dois índices para manter
Fusão RRF	Dispensa calibração de scores	Ignora a magnitude dos scores
Rerank com cross-encoder	Mais precisão nos primeiros resultados	Latência cresce com o nº de candidatos
Compressão de prompt	Até 20× menos tokens [23]	Chamada extra de modelo; pode perder detalhes
Orçamento de tokens	Custo previsível por requisição	Perda de recall se for baixo demais

8 Cache: de prompt, semântico e KV

O post cita "prompt caching" e "cache semântico" como alavancas separadas, e eles são frequentemente confundidos entre si e com o KV cache do motor de inferência. A Figura 6 os separa.

FIGURA 6

Três camadas de cache frequentemente confundidas

O que cada camada reutiliza e arrisca

Camada	Reutiliza	Regra de match	Economiza	Principal risco
Cache semântico Camada de aplicação (A4)	A resposta final inteira	Similaridade de embedding $\geq \tau$ dentro do escopo tenant/ACL/versão	O pipeline inteiro: custo e latência	Acertos falsos, respostas velhas, vazamento entre tenants
Prompt / prefix cache API do provedor (A3)	Prefixo de prompt já processado (estados de atenção)	Match exato de prefixo; ≥ 1.024 tokens (OpenAI), breakpoints explícitos (Anthropic)	Preço de entrada (leitura 0,1× na Anthropic) e tempo até o 1º token	Custo de escrita, TTL curto, canal lateral de tempo se compartilhado
KV cache Motor de inferência (self-hosted)	Tensores key/value dentro e entre requisições	Árvore de prefixos por token (RadixAttention), blocos paginados (PagedAttention)	Memória de GPU e throughput	Pressão de memória e política de despejo

Fontes: Bang 2023; Gill et al. 2024; Gim et al. 2024; Kwon et al. 2023; Zheng et al. 2025; docs Anthropic e OpenAI (2026).
© Aridio Silva @aridiosilva · Outubro 2026 · Pesquisador Independente · ORCID 0009-0008-2411-6995 · Figuras sob licença CC BY 4.0

Figura 6. Três camadas de cache. Elas reutilizam coisas diferentes, casam por regras diferentes e carregam riscos diferentes. © 2026 Aridio Silva (@aridiosilva), Pesquisador Independente, ORCID 0009-0008-2411-6995. CC BY 4.0.

A terceira camada só importa para equipes que hospedam seus próprios modelos. Os motores de inferência guardam na memória da GPU os tensores key-value (KV) de atenção de cada requisição. O PagedAttention os gerencia em blocos de tamanho fixo para reduzir a fragmentação [26], e o RadixAttention os compartilha entre requisições por meio

de uma árvore de prefixos [53]. O prompt caching do provedor é a forma gerenciada e precificada da mesma ideia.

8.1 A economia do prompt caching

Considere um prefixo de P tokens reutilizado por n requisições dentro do tempo de vida do cache, com os multiplicadores da Anthropic (escrita 1,25×, leitura 0,1×) [5]. O custo relativo do prefixo é:

$$\text{custo com cache} / \text{custo sem cache} = [1,25 + 0,1 \cdot (n - 1)] / n(3)$$

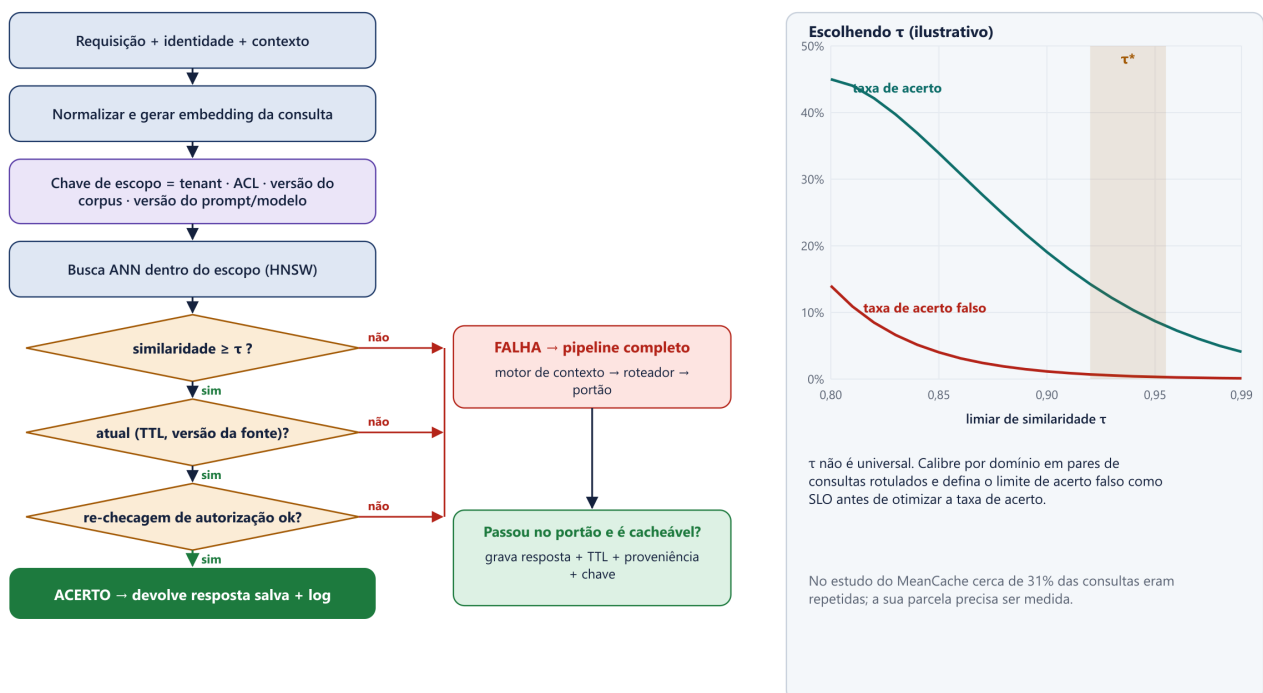
O cache se paga a partir do segundo uso (1,35 contra 2,0). Com $n = 10$, o prefixo custa 21,5% do preço sem cache. A alavanca tem baixo risco e alto retorno quando os prefixos são longos e o tráfego é constante. Só dá prejuízo quando um prefixo gravado nunca é reutilizado antes de expirar.

8.2 Cache semântico com segurança

FIGURA 7

Cache semântico: um acerto precisa ser similar, atual e autorizado

Fluxo de decisão e trade-off do limiar



Fontes: Bang 2023 (GPTCache); Gill et al. 2024 (MeanCache); Malkov & Yashunin 2016 (HNSW); Reimers & Gurevych 2019. Curvas ilustrativas.
© Aridio Silva @aridiosilva · Outubro 2026 · Pesquisador Independente · ORCID 0009-0008-2411-6995 · Figuras sob licença CC BY 4.0

Figura 7. Fluxo de decisão do cache semântico. Um acerto exige similaridade, atualidade e uma nova checagem de autorização. O gráfico do limiar é ilustrativo. © 2026 Aridio Silva (@aridiosilva), Pesquisador Independente, ORCID 0009-0008-2411-6995. CC BY 4.0.

O fluxo da Figura 7 acrescenta quatro controles que o infográfico apenas sugere ("autorizada, atual, semanticamente equivalente").

- Chave de escopo.** Buscar só entre entradas com o mesmo tenant, direitos de acesso, versão do corpus e versão do prompt ou modelo. Isso evita vazamento entre tenants, que a OWASP enquadra em divulgação de informação sensível e em fraquezas de vetores e embeddings [43].
- Limiar τ calibrado.** Elevar τ reduz os acertos falsos mais depressa do que reduz a taxa de acerto, até certo ponto. τ deve ser ajustado por domínio em pares rotulados [16].
- Atualidade.** Usar TTL mais invalidação quando muda a versão do documento de origem.
- Gravação só após o portão.** Guardar a proveniência junto com a resposta para que ela possa ser auditada ou expurgada.

Benefícios

- Um acerto evita o pipeline inteiro: a maior economia por evento.
- Menor latência em perguntas repetidas.

Riscos

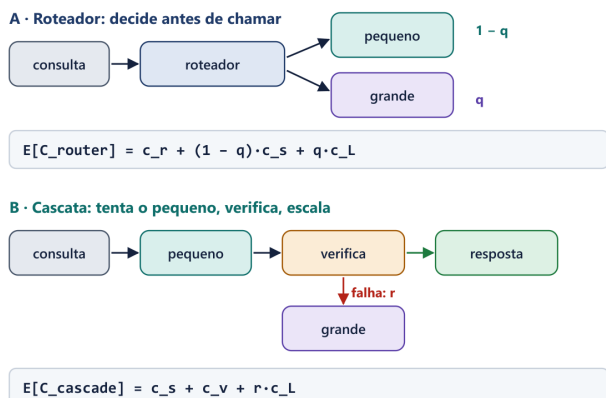
- Acertos falsos são respostas erradas ditas com confiança.
- Desatualização e vazamento entre tenants.
- Respostas personalizadas ou sensíveis ao tempo são más candidatas.

9 Roteamento de modelos e cascatas

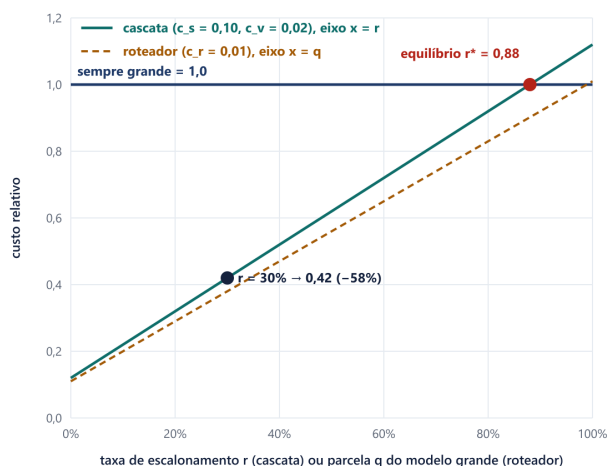
FIGURA 8

Roteamento vs. cascata, e quando a cascata deixa de compensar

Custo esperado relativo a usar sempre o modelo grande



A cascata só economiza enquanto $r < 1 - (c_s + c_v)/c_L$. Requisições escaladas também pagam a latência do modelo pequeno.



Fontes: Chen et al. 2023; Ding et al. 2024; Ong et al. 2024; Aggarwal et al. 2023; Yue et al. 2023; Gupta et al. 2024. Razões de custo ilustrativas. © Aridio Silva @aridiosilva · Outubro 2026 · Pesquisador Independente · ORCID 0009-0008-2411-6995 · Figuras sob licença CC BY 4.0

Figura 8. Roteador e cascata, seus custos esperados e a taxa de escalonamento de equilíbrio. Razões de custo ilustrativas. © 2026 Aridio Silva (@aridiosilva), Pesquisador Independente, ORCID 0009-0008-2411-6995. CC BY 4.0.

Sejam c_s , c_L , c_v e c_r os custos do modelo pequeno, do modelo grande, do verificador e do roteador. Um roteador envia uma parcela q do tráfego ao modelo grande. Uma cascata escala uma parcela r após a verificação.

$$E[C_{\text{roteador}}] = c_r + (1 - q) \cdot c_s + q \cdot c_L \quad E[C_{\text{cascata}}] = c_s + c_v + r \cdot c_L \quad (4)$$

A cascata supera "sempre grande" só enquanto $r < 1 - (c_s + c_v)/c_L$. Com $c_s = 0,10$ e $c_v = 0,02$ (relativos a c_L), o equilíbrio é $r^* = 0,88$. Com $r = 30\%$, a cascata custa $0,42$, uma economia de 58% . Isso é próximo da etapa de 40% de roteamento do post, uma vez incluído o nível médio.

Sinais de roteamento na literatura.

- Dificuldade prevista por um classificador treinado [13].
- Roteadores treinados com preferência humana [39].
- Consistência das respostas entre caminhos de raciocínio amostrados [51].
- Autoverificação com uma política POMDP [1].
- Incerteza por token, que corrige o viés de tamanho da confiança da sequência inteira [21].

Tabela 2. Roteador versus cascata

	Roteador (decide antes)	Cascata (tenta e depois escala)
Latência	Uma chamada de modelo mais um roteador pequeno	Requisições escaladas pagam duas chamadas em série
Qualidade do sinal	Precisa prever a dificuldade antes de ver a resposta	Julga uma resposta real, o que é mais confiável

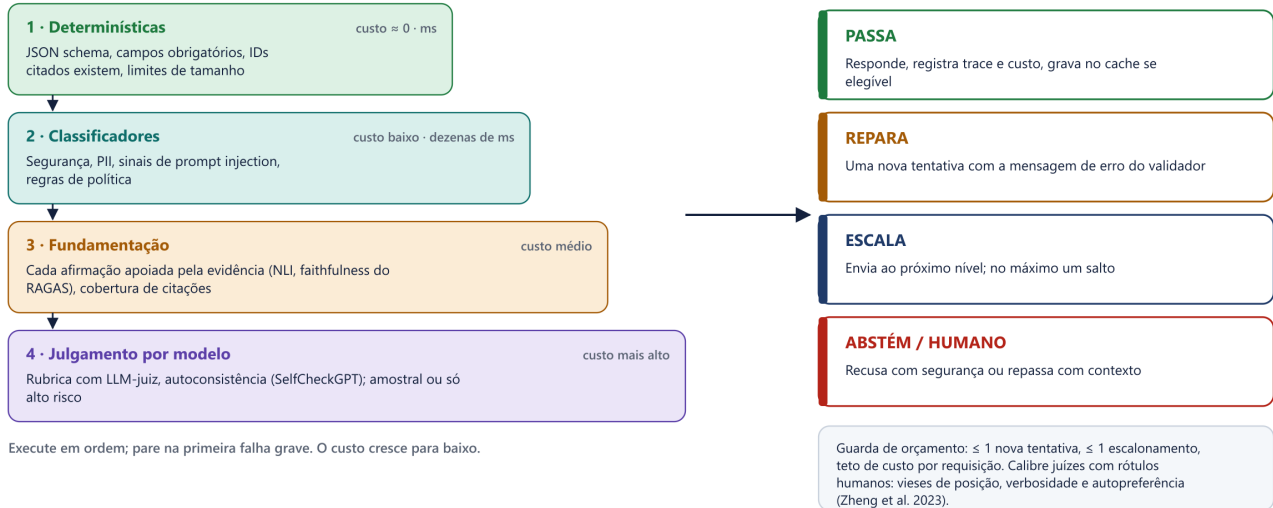
	Roteador (decide antes)	Cascata (tenta e depois escala)
Necessidade de dados	Dados rotulados ou de preferência para treinar	Um verificador ou checagem de consistência
Melhor para	Tráfego interativo, sensível à latência	Tráfego em lote ou sensível à acurácia

10 Portões de qualidade e escalonamento

FIGURA 9

Portão de qualidade: checagens mais baratas primeiro, escalonamento limitado

Camadas de checagem e desfechos



Execute em ordem; pare na primeira falha grave. O custo cresce para baixo.

Fontes: Es et al. 2023 (RAGAS); Manakul et al. 2023 (SelfCheckGPT); Zheng et al. 2023 (LLM-as-judge); Asai et al. 2023; Yan et al. 2024; OWASP 2025.
© Aridio Silva @aridiosilva · Outubro 2026 · Pesquisador Independente · ORCID 0009-0008-2411-6995 · Figuras sob licença CC BY 4.0

Figura 9. Portão de qualidade: quatro camadas de checagem, da mais barata à mais cara, com quatro desfechos possíveis e uma guarda de orçamento. © 2026 Aridio Silva (@aridiosilva), Pesquisador Independente, ORCID 0009-0008-2411-6995. CC BY 4.0.

O portão é o que torna seguro o roteamento agressivo. Ele deve rodar as checagens em ordem de custo e parar na primeira falha grave.

- 1. Checagens determinísticas.** JSON schema, campos obrigatórios e se cada ID citado de fato existe. São baratas e pegam as falhas mais frequentes de agentes.
- 2. Classificadores.** Sinais de segurança, PII e prompt injection (OWASP LLM01 e LLM02) [43].
- 3. Checagens de fundamentação.** Cada afirmação é apoiada pela evidência recuperada? O RAGAS define métricas sem referência, como fidelidade (faithfulness) e relevância do contexto [15, 44].
- 4. Julgamento por modelo.** O LLM-como-juiz chega a mais de 80% de concordância com preferências humanas, mas mostra vieses de posição, verbosidade e autopreferência [52]. O SelfCheckGPT detecta alucinações checando a consistência entre amostras [32]. Como essas checagens custam tanto quanto uma geração, devem rodar por amostragem ou só nas rotas de alto risco.

Desfechos. Cada resposta termina em um de quatro estados:

- **passa:** responde, registra e possivelmente grava no cache;
- **repara:** uma nova tentativa com a mensagem de erro do validador;
- **escala:** envia ao próximo nível de modelo, com no máximo um salto;
- **abstém ou repassa:** recusa com segurança ou repassa a um humano.

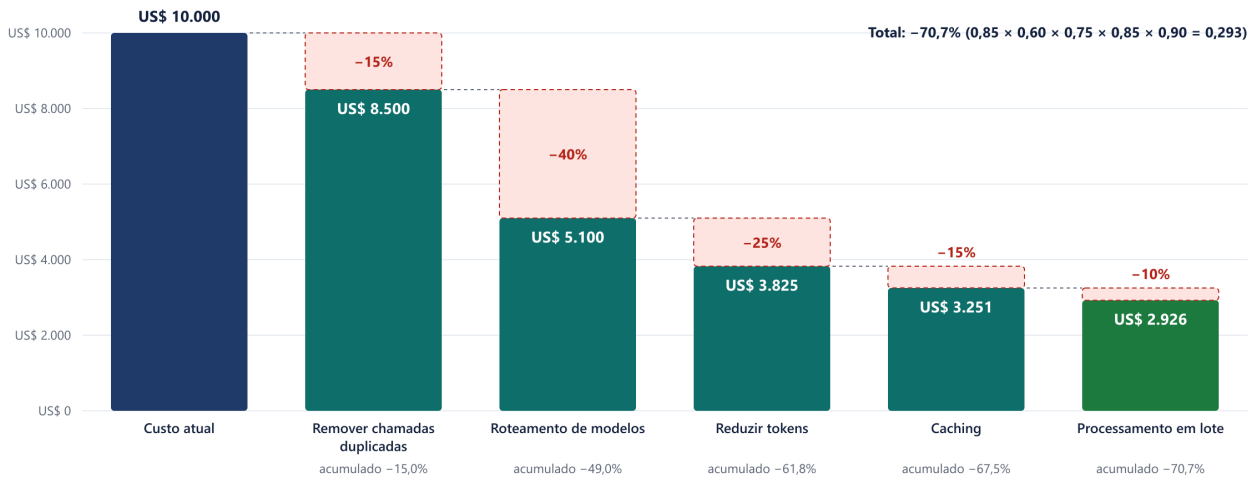
Uma *guarda de orçamento* limita novas tentativas, escalonamentos e gasto por requisição, o que trata o risco de "consumo ilimitado" da OWASP (LLM10) [43].

11 Decomposição do custo e sensibilidade

FIGURA 10

Decomposição do custo: cinco reduções sequenciais (redesenhado em escala)

US\$ por mês; valores do post



Baseado em R. Bhardwaj (2026), post no LinkedIn, redesenhado e ampliado. Percentuais por etapa calculados pelo autor; o post rotula o gráfico como ilustrativo.
© Aridio Silva @aridiosilva · Outubro 2026 · Pesquisador Independente · ORCID 0009-0008-2411-6995 · Figuras sob licença CC BY 4.0

Figura 10. A cascata do post, redesenhada em escala. Os percentuais por etapa e acumulados foram calculados pelo autor a partir dos valores do post. © 2026 Aridio Silva (@aridiosilva), Pesquisador Independente, ORCID 0009-0008-2411-6995. CC BY 4.0.

11.1 Conferindo a aritmética

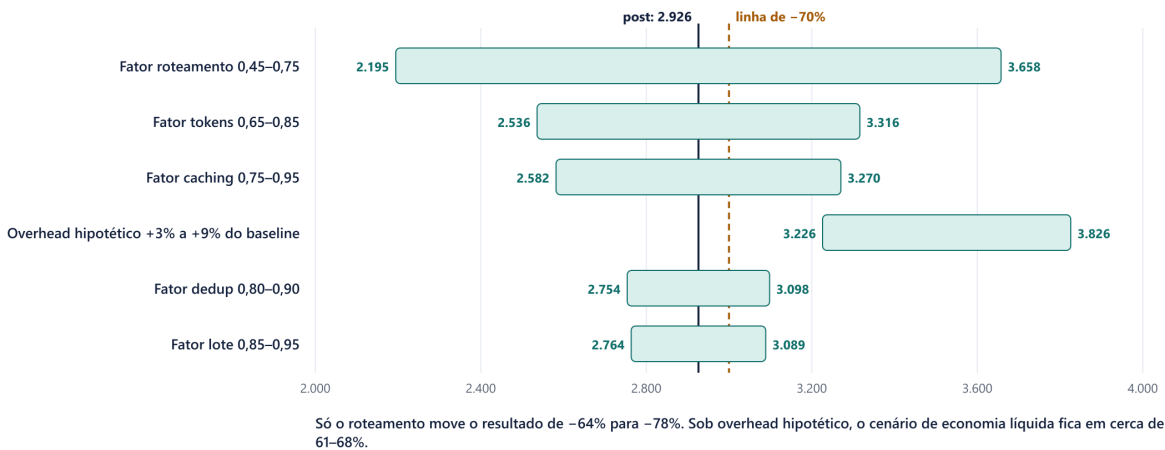
Os valores do post são internamente consistentes: $10.000 \times 0,85 \times 0,60 \times 0,75 \times 0,85 \times 0,90 = 2.926$. O roteamento é a maior etapa isolada (-40% do que resta; 34 dos 70,7 pontos percentuais). O post rotula o gráfico como *ilustrativo*, então esses números são metas de projeto, não medições.

11.2 Sensibilidade

FIGURA 11

Sensibilidade: quão robusta é a afirmação de 70%?

Custo mensal final, variando uma premissa por vez



Modelo e faixas do autor (sensibilidade uma-a-uma em torno dos fatores do post). Ilustrativo, não medido.
© Aridio Silva @aridiosilva · Outubro 2026 · Pesquisador Independente · ORCID 0009-0008-2411-6995 · Figuras sob licença CC BY 4.0

Figura 11. Sensibilidade uma-a-uma do custo mensal final. As faixas são premissas do autor em torno dos fatores de cada etapa do post. As cinco linhas verde-azuladas variam esses fatores para mais e para menos; a linha âmbar não é uma variação, e sim o overhead que as alavancas acrescentam, por isso só aumenta o custo. © 2026 Aridio Silva (@aridiosilva), Pesquisador Independente, ORCID 0009-0008-2411-6995. CC BY 4.0.

Variar cada fator dentro de uma faixa hipotética escolhida pelo autor mostra, nesse cenário, que o roteamento domina o resultado (de -64% a -78%). Lote e deduplicação são os que menos pesam. O post omite o custo extra: embeddings e o banco vetorial do cache semântico, o roteador, as chamadas de juiz e as rodadas de avaliação. Se esse custo extra somar, por hipótese, 3–9% do baseline original, a **economia líquida do cenário cai para cerca de 61–68%**. O cálculo demonstra sensibilidade, não uma faixa esperada para cargas reais; essa faixa depende de medição por workload.

Tabela 3. Custos que as alavancas acrescentam (não aparecem no post)

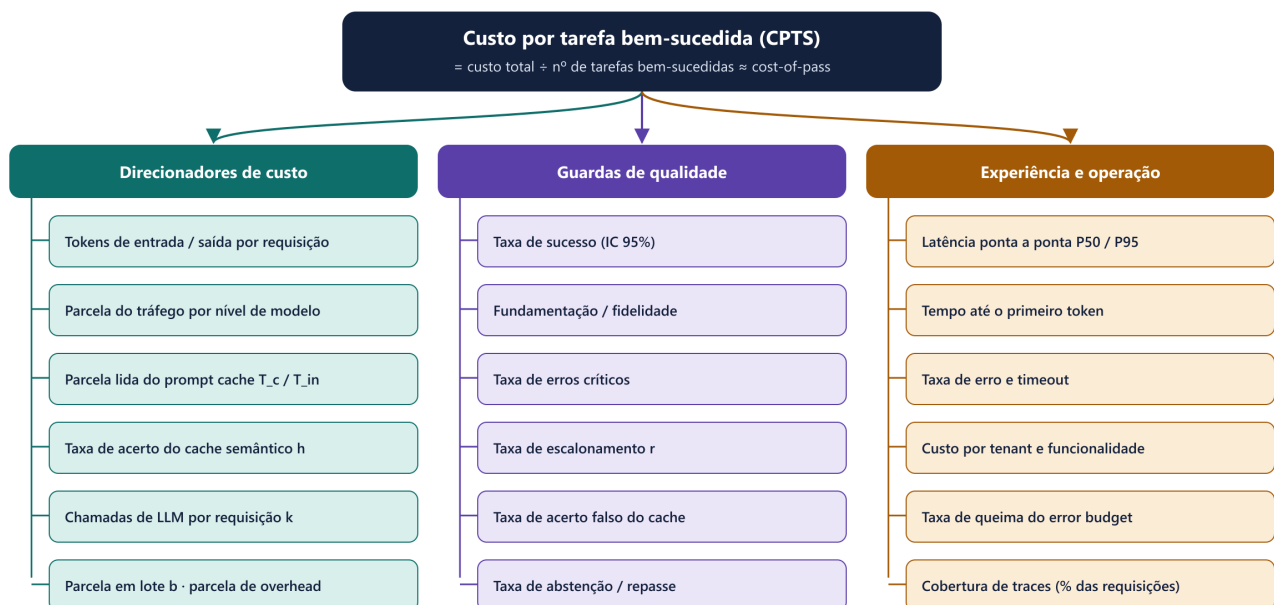
Item	Introduzido por	Como mantê-lo pequeno
Embeddings de consulta, busca vetorial	A4, A2	Modelo de embedding pequeno; cachear embeddings
Inferência do roteador	A1	Classificador leve, não um LLM
Chamadas de verificador e juiz	Portão	Checagens determinísticas primeiro; juízes por amostragem
Escritas no cache (1,25×–2×)	A3	Cachear só prefixos que serão reutilizados
Rodadas de avaliação e rotulagem	Todas	Rodar avaliações pela API de lote (-50%)
Tempo de engenharia e operação	Todas	Fasear o trabalho; ganhos rápidos primeiro (Seção 15)

12 Avaliação e KPIs

FIGURA 12

Árvore de KPIs: uma métrica norte, três famílias de direcionadores

O que medir antes e depois



Fontes: Erol et al. 2025 (cost-of-pass); Miller 2024 (barras de erro); Liang et al. 2022 (HELM); Beyer et al. 2016 (SLOs); OpenTelemetry GenAI.
© Aridio Silva @aridiosilva · Outubro 2026 · Pesquisador Independente · ORCID 0009-0008-2411-6995 · Figuras sob licença CC BY 4.0

Figura 12. Árvore de KPIs. O custo por tarefa bem-sucedida é a métrica norte; direcionadores de custo, guardas de qualidade e métricas operacionais explicam seu movimento. © 2026 Aridio Silva (@aridiosilva), Pesquisador Independente, ORCID 0009-0008-2411-6995. CC BY 4.0.

12.1 Lendo a tabela de qualidade com olhar crítico

O post reporta sucesso da tarefa 92% → 92–93%, fundamentação 90% → 90–91% e erros críticos 0,5% → 0,4%. Miller recomenda tratar os itens de avaliação como uma amostra de uma população maior e reportar intervalos de confiança [35]. Com 500 itens, o erro-padrão de uma taxa de sucesso de 92% é $\sqrt{(0,92 \cdot 0,08 / 500)} \approx 1,2$ ponto, então o intervalo de 95% é de cerca de $\pm 2,4$ pontos. As variações reportadas estão dentro desse ruído. "Mesma qualidade" é plausível, mas não está demonstrada.

Uma versão rigorosa deveria:

- avaliar os dois sistemas nos *mesmos* itens (desenho pareado);
- pré-registrar uma margem de não inferioridade, por exemplo "sucesso no máximo 1 ponto menor";
- dimensionar o conjunto de avaliação para que o intervalo fique mais estreito que essa margem.

12.2 Definição dos KPIs

Tabela 4. KPIs recomendados

KPI	Definição	Por que importa
Custo por tarefa bem-sucedida	Custo total ÷ tarefas bem-sucedidas [14]	Métrica norte; impede que o "barato mas errado" pareça bom
Tokens por requisição (entrada/saída)	Média e P95, por rota	Revela inchaço de contexto e saídas verbosas
Mix de níveis de modelo	Parcela das requisições por nível	Principal direcionador de p(m)
Parcela lida do prompt cache	Tokens de entrada em cache ÷ total	Confirma que a Alavanca 3 funciona
Taxas de acerto e de acerto falso do cache semântico	Acertos ÷ buscas; acertos errados ÷ acertos auditados	Economia e segurança da Alavanca 4
Taxa de escalonamento r	Escaladas ÷ requisições que passaram pelo portão	Equilíbrio da cascata (Eq. 4)
Sucesso da tarefa, fundamentação, erros críticos	Em conjunto fixo de avaliação, com IC 95% [35]	Guardas de qualidade
Latência P50/P95 e TTFT	Ponta a ponta e por etapa	Experiência do usuário; SLOs de latência [8]
Chamadas por requisição k	Chamadas de LLM ÷ requisições	Eficiência do loop agêntico (Alavanca 5)

13 Observabilidade e governança

Nenhuma das alavancas pode ser ajustada sem telemetria por requisição. As convenções semânticas do OpenTelemetry para IA generativa definem atributos de span padronizados para modelo, uso de tokens e operações [42]. Usá-las torna o custo por span portátil entre fornecedores.

A governança tem três âncoras:

- o NIST AI Risk Management Framework, com suas funções Govern, Map, Measure e Manage [36];
- o Perfil de IA Generativa do NIST [37];
- o OWASP Top 10 for LLM Applications 2025, que nomeia os principais riscos que esta arquitetura precisa controlar: prompt injection, divulgação de informação sensível, fraquezas de vetores e embeddings e consumo ilimitado [43].

O alerta de Sculley et al. sobre a dívida técnica oculta em sistemas de ML se aplica diretamente [47]. Cada cache, roteador e juiz é mais um componente cuja configuração pode se desviar. Cada um precisa de um responsável, uma métrica e um alerta.

14 Relação com o SGAEIA: eficiência sob governança e evidência

Classificação desta seção: SGAEIA INTERPRETATION — conexão conceitual pública, não arquitetura normativa nem evidência de implementação.

Para o SGAEIA — Secure Governed Autonomous Edge Intelligence Architecture — reduzir custo não é um objetivo isolado. Uma otimização só é aceitável quando preserva limites de autoridade, aplicação de políticas, rastreabilidade, revogação, separação entre tenants e evidência suficiente para explicar por que cada requisição seguiu determinada rota. Por isso, a métrica de custo por tarefa bem-sucedida deve ser acompanhada por erros críticos, falsos positivos e negativos dos portões, latência por rota, completude dos traces e risco residual; uma resposta mais barata que

enfraquece esses controles não representa ganho líquido para um sistema autônomo governado [56].

As seis alavancas analisadas se relacionam principalmente com quatro planos de responsabilidade do projeto. O roteamento e as cascatas pertencem à integração e substituição: trocar o modelo não pode ampliar silenciosamente a autoridade disponível nem invalidar pressupostos de segurança. Os caches e o motor de contexto atravessam identidade e confiança, autoridade e delegação, porque uma resposta reutilizada precisa respeitar o tenant, a autorização atual, a proveniência e a versão do corpus. Os limites de passos, tentativas e gastos integram a execução governada, enquanto telemetria por requisição, decisões de rota e resultados dos portões integram observabilidade e evidência.

Dessa conexão decorre uma propriedade candidata, ainda não normativa: **a otimização de custo de um sistema de AI não deve degradar as fronteiras de autoridade, a aplicação de políticas, a capacidade de revogação nem a completude da evidência necessária à auditoria**. A arquitetura de seis alavancas deste relatório é evidência externa e um padrão de engenharia possível; ela não se torna componente do SGAEIA por aparecer nesta publicação. Qualquer adoção exigiria auditoria de cobertura existente, análise de impacto, passagem pelos gates Research-to-Architecture aplicáveis, critérios de aceitação e verificação adversarial antes de uma decisão do proprietário do projeto.

Essa leitura também altera a ordem prática das alavancas. Instrumentação, eliminação de chamadas redundantes, limites de loop e prompt caching oferecem ganhos com menor superfície de risco. Roteamento, juízes e cache semântico exigem avaliação de não inferioridade, testes de vazamento, análise de falsos acertos, verificação de substituição de modelo e modo de falha seguro. O objetivo não é obter a menor fatura possível, mas o menor custo compatível com execução governada e assurance contínua sustentada por evidências.

15 Roteiro de implementação

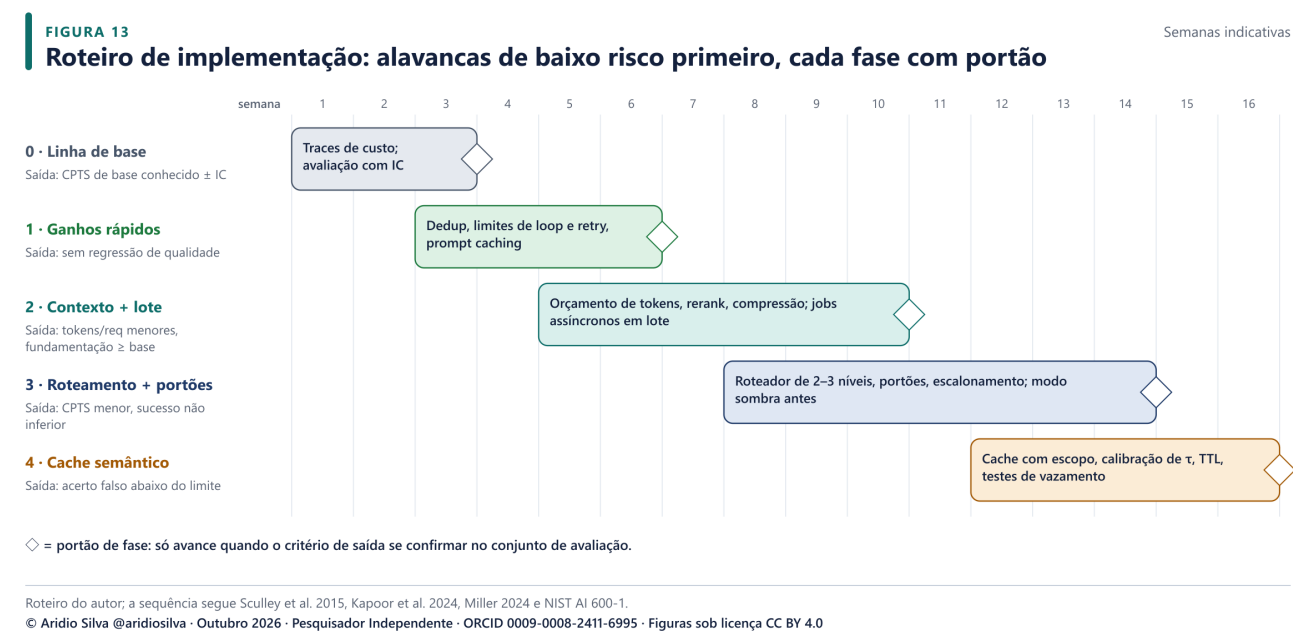


Figura 13. Roteiro em fases. Cada fase termina em um portão cujo critério de saída precisa se confirmar no conjunto de avaliação antes de a próxima começar. Semanas indicativas. © 2026 Aridio Silva (@aridiosilva), Pesquisador Independente, ORCID 0009-0008-2411-6995. CC BY 4.0.

Tabela 5. Fases, ações e fontes de apoio

Fase	Ações	Critério de saída	Apoio
0 - Linha de base	Registrar cada chamada com tokens e custo; montar um conjunto de avaliação com intervalos de confiança	Custo de base por tarefa bem-sucedida conhecido $\pm$ IC	[14, 35, 42]

Fase	Ações	Critério de saída	Apoio
1 · Ganhos rápidos	Remover chamadas duplicadas; limitar loops e novas tentativas; reestruturar prompts para cache de prefixo	Sem regressão de qualidade; parcela lida do cache na meta	[2, 5, 25, 41]
2 · Contexto e lote	Busca híbrida, rerank, orçamento de tokens, compressão; levar jobs assíncronos para lote	Tokens por requisição menores; fundamentação $\geq$ base	[12, 24, 29, 38, 40]
3 · Roteamento e portões	Roteador de 2–3 níveis; camadas do portão; escalonamento limitado; modo sombra antes da virada	Custo por tarefa bem-sucedida menor; sucesso não inferior	[13, 15, 39, 51, 52]
4 · Cache semântico	Cache com escopo; calibração de τ em pares rotulados; TTL e invalidação; testes de vazamento	Taxa de acerto falso abaixo do limite acordado	[7, 16, 43]

A ordem inverte de propósito a ordem visual do post: as alavancas de menor risco e maior certeza vêm primeiro. O cache semântico fica por último porque é a única alavanca capaz de devolver uma resposta errada sem nenhum modelo envolvido, por isso exige a governança mais madura.

16 Discussão e ameaças à validade

- **Dependência da carga.** Todos os números do post dependem do mix de requisições. Uma carga de tarefas difíceis, únicas e com várias etapas tem pouco a rotear ou cachear.
- **Volatilidade de preços.** Os preços e multiplicadores de cache dos provedores mudam várias vezes por ano, e a diferença entre níveis de modelo pode diminuir [14]. A arquitetura continua válida; o tamanho da economia não.
- **Afirmção sobre latência.** O post reporta latência P95 menor (2,8 s $\rightarrow$ 2,6 s). É plausível se os acertos de cache e os modelos menores compensarem as etapas extras, mas cascatas acrescentam chamadas em série nas requisições escaladas. Isso precisa ser medido por rota.
- **Validade da avaliação.** Benchmarks de agentes muitas vezes não têm conjuntos de holdout adequados [25]. Roteadores ajustados no próprio conjunto de avaliação vão superestimar os ganhos.
- **Segurança dos caches.** Os dois tipos de cache criam canais de informação entre usuários se forem compartilhados [20, 43].
- **Escopo deste relatório.** É uma análise fundamentada na literatura, não um experimento. Suas ampliações numéricas (Figuras 8 e 11) são modelos ilustrativos.

17 Conclusão

A proposta de Bhardwaj é tecnicamente sólida e condiz com as evidências publicadas. Cada uma das seis alavancas tem apoio em fontes primárias, e vários resultados publicados superam as reduções que ela assume. Sua principal contribuição é um princípio: gastar inteligência onde a tarefa precisa, e medir o resultado como custo por tarefa bem-sucedida. Sua principal fragilidade é o número da manchete. Ele é ilustrativo, omite o custo extra que as alavancas trazem, e sua comparação de qualidade carece de apoio estatístico.

Acrescentadas essas peças, o cenário modelado produz economia líquida de 61–68% sob as premissas escolhidas, mas não estabelece uma faixa esperada para produção. Demonstrar uma redução próxima de 70% exige telemetria de workload, custos completos e um teste de não inferioridade pareado. O caminho mais seguro é o da Seção 15: medir primeiro, colher as alavancas de baixo risco e só então introduzir roteamento e cache semântico, sempre atrás de portões de qualidade e de um teste de não inferioridade.

Glossário

Acerto falso - Um acerto do cache semântico que devolve a resposta de uma pergunta diferente.

Alucinação - Saída fluente que não é apoiada pela entrada nem pelos fatos.

ANN (vizinho mais próximo aproximado) - Busca que encontra rapidamente vetores próximos ao vetor da consulta, ao custo de às vezes não achar exatamente o mais próximo.

API de lote (Batch API) - Endpoint assíncrono que processa muitas requisições dentro de uma janela (24 h) com desconto (50% nos grandes provedores).

BM25 - Função de ranqueamento lexical baseada em frequência do termo, frequência inversa nos documentos e tamanho do documento.

Busca densa / embedding - Representar textos como vetores de modo que textos de sentido parecido fiquem próximos, e então buscar por similaridade vetorial.

Busca híbrida - Rodar busca lexical (BM25) e densa em conjunto e fundir os resultados.

Cache semântico - Cache na aplicação que devolve uma resposta salva quando uma nova consulta tem significado suficientemente parecido com uma anterior.

Cascata - Sequência de modelos do mais barato ao mais caro. A requisição só passa ao próximo modelo quando uma checagem da resposta atual falha.

Compressão de prompt - Remover tokens de pouca informação de um prompt com um modelo menor, preservando o sentido.

Cost-of-pass / custo por tarefa bem-sucedida - Gasto esperado para obter um resultado correto: custo por tentativa dividido pela probabilidade de sucesso.

Cross-encoder - Modelo que lê consulta e trecho juntos para pontuar a relevância. Mais preciso e mais lento que comparar embeddings separados.

Engenharia de contexto - Escolher quais tokens entram na janela de contexto do modelo, e em que ordem, em cada passo.

Escalonamento - Reenviar uma requisição a um modelo mais capaz depois que uma checagem de qualidade falha.

Fronteira de Pareto - O conjunto de opções em que nenhuma é melhor num objetivo sem ser pior em outro.

Fundamentação / fidelidade - O grau em que cada afirmação da resposta é apoiada pela evidência entregue ao modelo.

HNSW - Hierarchical navigable small world: índice muito usado para busca rápida de vizinhos mais próximos aproximados.

Intervalo de confiança (IC) - Faixa que, em amostragens repetidas, contém o valor verdadeiro na taxa declarada (por exemplo, 95%).

Janela de contexto - O número máximo de tokens que um modelo lê numa chamada, incluindo instruções, evidências e histórico.

KV cache - Os tensores key e value da atenção que o motor de inferência mantém em memória para não recalcular tokens anteriores.

LLM-como-juiz - Usar um modelo de linguagem para avaliar a saída de outro modelo segundo uma rubrica.

Loop agêntico - O ciclo em que um agente raciocina, chama uma ferramenta, lê o resultado e decide o próximo passo. Cada passo é uma chamada paga ao modelo.

Lost in the middle - A constatação de que os modelos usam pior a informação do meio de um contexto longo do que a do início ou do fim.

Modelo de fronteira - O nível de modelo mais capaz, e geralmente mais caro, disponível num dado momento.

Modo sombra - Rodar um novo componente sobre o tráfego real sem usar sua saída, para compará-lo com o atual.

Observabilidade / trace / span - Registrar cada requisição como um trace composto de spans (um por etapa), com atributos como tokens, custo e latência.

Orçamento de tokens - Um teto fixo de tokens que uma etapa pode enviar ou gerar.

Prompt caching (cache de prefixo) - Reuso, no provedor, do estado já processado de um prefixo de prompt que se repete entre requisições, cobrado com desconto.

Prompt injection - Entrada elaborada para sobrepor as instruções do modelo. O principal risco da OWASP para aplicações com LLM.

RAG (geração aumentada por recuperação) - Recuperar documentos relevantes e incluí-los no prompt para que o modelo responda a partir deles.

Regra de adiamento (deferral) - A regra que a cascata usa para decidir se mantém a resposta ou passa a requisição a um modelo mais forte.

Rerank (reordenação) - Uma segunda pontuação, mais precisa, sobre os melhores candidatos de um recuperador de primeira etapa.

Roteamento de modelos - Escolher, por requisição, qual modelo deve respondê-la, antes de chamar qualquer modelo.

RRF (reciprocal rank fusion) - Fundir listas ranqueadas somando $1/(k + \text{posição})$ para cada documento.

SLO / error budget - Uma meta para uma métrica do serviço e a quantidade de falha tolerada antes de exigir ação.

Teste de não inferioridade - Teste que mostra que o novo sistema não é pior que o antigo por mais que uma margem pré-definida.

Token - A unidade de texto que o modelo lê e escreve, e a unidade de cobrança. Cerca de $\frac{3}{4}$ de uma palavra em inglês.

TTFT (tempo até o primeiro token) - Latência entre enviar a requisição e receber o primeiro token de saída.

TTL (tempo de vida) - Quanto tempo uma entrada de cache permanece válida antes de expirar.

Referências (auditadas)

Todos os links foram resolvidos em 27 de outubro de 2026. Os registros do arXiv foram conferidos na API do arXiv (título, autores, data). Os demais links foram acessados e seu status HTTP registrado. Quando a editora bloqueia acesso automatizado (HTTP 403), confirmou-se que o DOI resolve para o registro da editora. O veículo de publicação é informado só quando confirmado; caso contrário, cita-se a versão do arXiv.

1. Aggarwal, P., Madaan, A., Anand, A., Potharaju, S. P., et al. (2023). *AutoMix: Automatically mixing language models*. arXiv preprint arXiv:2310.12963. <https://arxiv.org/abs/2310.12963>Verificado: metadados conferidos na API do arXiv, HTTP 200 · Acesso em 2026-10-27
2. Anthropic (2024, dezembro 19). *Building effective AI agents*. Anthropic Engineering. <https://www.anthropic.com/research/building-effective-agents>Verificado: HTTP 200 na URL atual indicada · Acesso em 2026-10-27
3. Anthropic (2025, setembro 29). *Effective context engineering for AI agents*. Anthropic Engineering. <https://www.anthropic.com/research/effective-context-engineering-for-ai-agents>Verificado: HTTP 200 na URL atual indicada · Acesso em 2026-10-27
4. Anthropic (2026). *Batch processing*. Claude Developer Platform documentation. <https://platform.claude.com/docs/en/build-with-claude/batch-processing>Verificado: HTTP 200 na URL atual indicada · Acesso em 2026-10-27
5. Anthropic (2026). *Prompt caching*. Claude Developer Platform documentation. <https://platform.claude.com/docs/en/build-with-claude/prompt-caching>Verificado: HTTP 200 na URL atual indicada · Acesso em 2026-10-27
6. Asai, A., Wu, Z., Wang, Y., Sil, A., & Hajishirzi, H. (2023). *Self-RAG: Learning to retrieve, generate, and critique through self-reflection*. ICLR 2024. arXiv:2310.11511. <https://arxiv.org/abs/2310.11511>Verificado: metadados conferidos na API do arXiv, HTTP 200 · Acesso em 2026-10-27
7. Bang, F. (2023). *GPTCache: An open-source semantic cache for LLM applications enabling faster answers and cost savings*. Proceedings of the 3rd Workshop for Natural Language Processing Open Source Software (NLP-OSS 2023), ACL. <https://aclanthology.org/2023.nlposs-1.24/>Verificado: HTTP 200, título conferido · Acesso em 2026-10-27
8. Beyer, B., Jones, C., Petoff, J., & Murphy, N. R. (Eds.) (2016). *Site reliability engineering: How Google runs production systems. Chapter 4: Service level objectives*. O'Reilly Media. <https://sre.google/sre-book/service-level-objectives/>Verificado: HTTP 200, título conferido · Acesso em 2026-10-27
9. Bhardwaj, R. (2026, outubro 26). *Cut LLM costs by up to 70%, not by compromising intelligence, but by engineering smarter model routing, context, caching, and agentic workflows [Post with infographic]*. LinkedIn. https://www.linkedin.com/posts/rajeev-bhardwaj_genai-agenticai-llmoptimization-share-7520411441674387456-qL-QVerificado: HTTP 200, título conferido · Acesso em 2026-10-27
10. Chen, L., Zaharia, M., & Zou, J. (2023). *FrugalGPT: How to use large language models while reducing cost and improving performance*. arXiv preprint arXiv:2305.05176. <https://arxiv.org/abs/2305.05176>Verificado: metadados conferidos na API do arXiv, HTTP 200 · Acesso em 2026-10-27
11. Chroma Research (2025). *Context rot: How increasing input tokens impacts LLM performance*. Chroma technical report. <https://www.trychroma.com/research/context-rot>Verificado: HTTP 200 na URL atual indicada · Acesso em

12. Cormack, G. V., Clarke, C. L. A., & Büttcher, S. (2009). *Reciprocal rank fusion outperforms Condorcet and individual rank learning methods*. Proceedings of the 32nd ACM SIGIR Conference, 758–759.
<https://doi.org/10.1145/1571941.1572114>Verificado: DOI resolve; a editora bloqueia acesso automatizado (HTTP 403) · Acesso em 2026-10-27
13. Ding, D., Mallick, A., Wang, C., Sim, R., et al. (2024). *Hybrid LLM: Cost-efficient and quality-aware query routing*. ICLR 2024. arXiv:2404.14618. <https://arxiv.org/abs/2404.14618>Verificado: metadados conferidos na API do arXiv, HTTP 200 · Acesso em 2026-10-27
14. Erol, M. H., El, B., Suzgun, M., Yuksekgonul, M., et al. (2025). *Cost-of-pass: An economic framework for evaluating language models*. arXiv preprint arXiv:2504.13359. <https://arxiv.org/abs/2504.13359>Verificado: metadados conferidos na API do arXiv, HTTP 200 · Acesso em 2026-10-27
15. Es, S., James, J., Espinosa-Anke, L., & Schockaert, S. (2023). *RAGAS: Automated evaluation of retrieval augmented generation*. EACL 2024 System Demonstrations. arXiv:2309.15217. <https://arxiv.org/abs/2309.15217>Verificado: metadados conferidos na API do arXiv, HTTP 200 · Acesso em 2026-10-27
16. Gill, W., Elidrisi, M., Kalapatapu, P., Ahmed, A., et al. (2024). *MeanCache: User-centric semantic caching for LLM web services*. arXiv preprint arXiv:2403.02694. <https://arxiv.org/abs/2403.02694>Verificado: metadados conferidos na API do arXiv, HTTP 200 · Acesso em 2026-10-27
17. Gim, I., Chen, G., Lee, S., Sarda, N., et al. (2023). *Prompt cache: Modular attention reuse for low-latency inference*. MLSys 2024. arXiv:2311.04934. <https://arxiv.org/abs/2311.04934>Verificado: metadados conferidos na API do arXiv, HTTP 200 · Acesso em 2026-10-27
18. Google (2026). *Batch API*. Gemini API documentation. <https://ai.google.dev/gemini-api/docs/batch-api>Verificado: HTTP 200 na URL atual indicada · Acesso em 2026-10-27
19. Google (2026). *Context caching*. Gemini API documentation. <https://ai.google.dev/gemini-api/docs/caching>Verificado: HTTP 200, título conferido · Acesso em 2026-10-27
20. Gu, C., Li, X. L., Kudritpudi, R., Liang, P., & Hashimoto, T. (2025). *Auditing prompt caching in language model APIs*. arXiv preprint arXiv:2502.07776. <https://arxiv.org/abs/2502.07776>Verificado: metadados conferidos na API do arXiv, HTTP 200 · Acesso em 2026-10-27
21. Gupta, N., Narasimhan, H., Jitkrittum, W., Rawat, A. S., et al. (2024). *Language model cascades: Token-level uncertainty and beyond*. arXiv preprint arXiv:2404.10136. <https://arxiv.org/abs/2404.10136>Verificado: metadados conferidos na API do arXiv, HTTP 200 · Acesso em 2026-10-27
22. Han, T., Wang, Z., Fang, C., Zhao, S., et al. (2024). *Token-budget-aware LLM reasoning*. arXiv preprint arXiv:2412.18547. <https://arxiv.org/abs/2412.18547>Verificado: metadados conferidos na API do arXiv, HTTP 200 · Acesso em 2026-10-27
23. Jiang, H., Wu, Q., Lin, C.-Y., Yang, Y., & Qiu, L. (2023). *LLMLingua: Compressing prompts for accelerated inference of large language models*. EMNLP 2023. arXiv:2310.05736. <https://arxiv.org/abs/2310.05736>Verificado: metadados conferidos na API do arXiv, HTTP 200 · Acesso em 2026-10-27

24. Jiang, H., Wu, Q., Luo, X., Li, D., et al. (2023). *LongLLMLingua: Accelerating and enhancing LLMs in long context scenarios via prompt compression*. ACL 2024. arXiv:2310.06839. <https://arxiv.org/abs/2310.06839>Verificado: metadados conferidos na API do arXiv, HTTP 200 · Acesso em 2026-10-27
25. Kapoor, S., Stroebl, B., Siegel, Z. S., Nadgir, N., & Narayanan, A. (2024). *AI agents that matter*. arXiv preprint arXiv:2407.01502. <https://arxiv.org/abs/2407.01502>Verificado: metadados conferidos na API do arXiv, HTTP 200 · Acesso em 2026-10-27
26. Kwon, W., Li, Z., Zhuang, S., Sheng, Y., et al. (2023). *Efficient memory management for large language model serving with PagedAttention*. SOSP 2023. arXiv:2309.06180. <https://arxiv.org/abs/2309.06180>Verificado: metadados conferidos na API do arXiv, HTTP 200 · Acesso em 2026-10-27
27. Lewis, P., Perez, E., Piktus, A., Petroni, F., et al. (2020). *Retrieval-augmented generation for knowledge-intensive NLP tasks*. NeurIPS 2020. arXiv:2005.11401. <https://arxiv.org/abs/2005.11401>Verificado: metadados conferidos na API do arXiv, HTTP 200 · Acesso em 2026-10-27
28. Liang, P., Bommasani, R., Lee, T., Tsipras, D., et al. (2022). *Holistic evaluation of language models*. Transactions on Machine Learning Research. arXiv:2211.09110. <https://arxiv.org/abs/2211.09110>Verificado: metadados conferidos na API do arXiv, HTTP 200 · Acesso em 2026-10-27
29. Liu, N. F., Lin, K., Hewitt, J., Paranjape, A., et al. (2024). *Lost in the middle: How language models use long contexts*. Transactions of the ACL, 12. <https://doi.org/10.1162/tacla.00638>. <https://arxiv.org/abs/2307.03172>Verificado: metadados conferidos na API do arXiv, HTTP 200 · Acesso em 2026-10-27
30. LMSYS (2024–2026). *RouteLLM: A framework for serving and evaluating LLM routers [Software]*. GitHub. <https://github.com/lm-sys/RouteLLM>Verificado: HTTP 200, título conferido · Acesso em 2026-10-27
31. Malkov, Y. A., & Yashunin, D. A. (2016). *Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs*. IEEE TPAMI (2020). arXiv:1603.09320. <https://arxiv.org/abs/1603.09320>Verificado: metadados conferidos na API do arXiv, HTTP 200 · Acesso em 2026-10-27
32. Manakul, P., Liusie, A., & Gales, M. J. F. (2023). *SelfCheckGPT: Zero-resource black-box hallucination detection for generative large language models*. EMNLP 2023. arXiv:2303.08896. <https://arxiv.org/abs/2303.08896>Verificado: metadados conferidos na API do arXiv, HTTP 200 · Acesso em 2026-10-27
33. Miao, X., Oliaro, G., Zhang, Z., Cheng, X., et al. (2023). *Towards efficient generative large language model serving: A survey from algorithms to systems*. arXiv preprint arXiv:2312.15234. <https://arxiv.org/abs/2312.15234>Verificado: metadados conferidos na API do arXiv, HTTP 200 · Acesso em 2026-10-27
34. Microsoft (2023–2026). *LLMLingua [Software]*. GitHub. <https://github.com/microsoft/LLMLingua>Verificado: HTTP 200, título conferido · Acesso em 2026-10-27
35. Miller, E. (2024). *Adding error bars to evals: A statistical approach to language model evaluations*. arXiv preprint arXiv:2411.00640. <https://arxiv.org/abs/2411.00640>Verificado: metadados conferidos na API do arXiv, HTTP 200 · Acesso em 2026-10-27
36. National Institute of Standards and Technology (2023). *Artificial intelligence risk management framework (AI RMF 1.0)*. NIST AI 100-1. <https://doi.org/10.6028/NIST.AI.100-1>Verificado: HTTP 200 na URL atual indicada · Acesso em 2026-10-27

37. National Institute of Standards and Technology (2024). *Artificial intelligence risk management framework: Generative artificial intelligence profile*. NIST AI 600-1. <https://doi.org/10.6028/NIST.AI.600-1> Verificado: HTTP 200 na URL atual indicada · Acesso em 2026-10-27
38. Nogueira, R., & Cho, K. (2019). *Passage re-ranking with BERT*. arXiv preprint arXiv:1901.04085. <https://arxiv.org/abs/1901.04085> Verificado: metadados conferidos na API do arXiv, HTTP 200 · Acesso em 2026-10-27
39. Ong, I., Almahairi, A., Wu, V., Chiang, W.-L., et al. (2024). *RouteLLM: Learning to route LLMs with preference data*. arXiv preprint arXiv:2406.18665. <https://arxiv.org/abs/2406.18665> Verificado: metadados conferidos na API do arXiv, HTTP 200 · Acesso em 2026-10-27
40. OpenAI (2026). *Batch API*. OpenAI API documentation. <https://developers.openai.com/api/docs/guides/batch> Verificado: HTTP 200 na URL atual indicada · Acesso em 2026-10-27
41. OpenAI (2026). *Prompt caching*. OpenAI API documentation. <https://developers.openai.com/api/docs/guides/prompt-caching> Verificado: HTTP 200 na URL atual indicada · Acesso em 2026-10-27
42. OpenTelemetry (2026). *Semantic conventions for generative AI systems*. GitHub: open-telemetry/semantic-conventions-genai. <https://github.com/open-telemetry/semantic-conventions-genai> Verificado: HTTP 200, título conferido · Acesso em 2026-10-27
43. OWASP GenAI Security Project (2025). *OWASP Top 10 for LLM applications 2025*. OWASP Foundation. <https://genai.owasp.org/llm-top-10/> Verificado: HTTP 200, título conferido · Acesso em 2026-10-27
44. Ragas contributors (2023–2026). *Ragas: Evaluation toolkit for LLM applications [Software]*. GitHub. <https://github.com/vibrantlabsai/ragas> Verificado: HTTP 200 na URL atual indicada · Acesso em 2026-10-27
45. Reimers, N., & Gurevych, I. (2019). *Sentence-BERT: Sentence embeddings using Siamese BERT-networks*. EMNLP-IJCNLP 2019, ACL. <https://doi.org/10.18653/v1/D19-1410>. <https://aclanthology.org/D19-1410/> Verificado: HTTP 200 na URL atual indicada · Acesso em 2026-10-27
46. Robertson, S., & Zaragoza, H. (2009). *The probabilistic relevance framework: BM25 and beyond*. Foundations and Trends in Information Retrieval, 3(4), 333–389. <https://doi.org/10.1561/15000000019> Verificado: DOI resolve; a editora bloqueia acesso automatizado (HTTP 403) · Acesso em 2026-10-27
47. Sculley, D., Holt, G., Golovin, D., Davydov, E., et al. (2015). *Hidden technical debt in machine learning systems*. Advances in Neural Information Processing Systems 28. <https://proceedings.neurips.cc/paper/2015/hash/86df7dcfd896fcdf2674f757a2463eba-Abstract.html> Verificado: HTTP 200, título conferido · Acesso em 2026-10-27
48. Sun, W., Yan, L., Ma, X., Wang, S., et al. (2023). *Is ChatGPT good at search? Investigating large language models as re-ranking agents*. EMNLP 2023. arXiv:2304.09542. <https://arxiv.org/abs/2304.09542> Verificado: metadados conferidos na API do arXiv, HTTP 200 · Acesso em 2026-10-27
49. Yan, S.-Q., Gu, J.-C., Zhu, Y., & Ling, Z.-H. (2024). *Corrective retrieval augmented generation*. arXiv preprint arXiv:2401.15884. <https://arxiv.org/abs/2401.15884> Verificado: metadados conferidos na API do arXiv, HTTP 200 ·

50. Yao, S., Zhao, J., Yu, D., Du, N., et al. (2022). *ReAct: Synergizing reasoning and acting in language models*. ICLR 2023. arXiv:2210.03629. <https://arxiv.org/abs/2210.03629>Verificado: metadados conferidos na API do arXiv, HTTP 200 · Acesso em 2026-10-27
51. Yue, M., Zhao, J., Zhang, M., Du, L., & Yao, Z. (2023). *Large language model cascades with mixture of thoughts representations for cost-efficient reasoning*. ICLR 2024. arXiv:2310.03094. <https://arxiv.org/abs/2310.03094>Verificado: metadados conferidos na API do arXiv, HTTP 200 · Acesso em 2026-10-27
52. Zheng, L., Chiang, W.-L., Sheng, Y., Zhuang, S., et al. (2023). *Judging LLM-as-a-judge with MT-Bench and Chatbot Arena*. NeurIPS 2023 Datasets and Benchmarks. arXiv:2306.05685. <https://arxiv.org/abs/2306.05685>Verificado: metadados conferidos na API do arXiv, HTTP 200 · Acesso em 2026-10-27
53. Zheng, L., Yin, L., Xie, Z., Sun, C., et al. (2024). *SGLang: Efficient execution of structured language model programs*. NeurIPS 2024. arXiv:2312.07104. <https://arxiv.org/abs/2312.07104>Verificado: metadados conferidos na API do arXiv, HTTP 200 · Acesso em 2026-10-27
54. Zhou, Z., Ning, X., Hong, K., Fu, T., et al. (2024). *A survey on efficient inference for large language models*. arXiv preprint arXiv:2404.14294. <https://arxiv.org/abs/2404.14294>Verificado: metadados conferidos na API do arXiv, HTTP 200 · Acesso em 2026-10-27
55. Zilliz (2023–2026). *GPTCache: Semantic cache for LLMs [Software]*. GitHub. <https://github.com/zilliztech/GPTCache>Verificado: HTTP 200, título conferido · Acesso em 2026-10-27
56. Silva, Aridio. (2026). *SGAEIA: Secure Governed Autonomous Edge Intelligence Architecture* (Version 0.3.4) [Research software artifact]. Zenodo. <https://doi.org/10.5281/zenodo.22557796>

Como citar

Silva, Aridio. (2026). *Como Reduzir Custos de Sistemas de AI com LLMs sem Sacrificar Qualidade: Uma revisão baseada em evidências de seis alavancas, seus limites e uma arquitetura de referência governada* (Versão 1.1) [Relatório técnico]. Pesquisa independente. ORCID 0009-0008-2411-6995. DOI: <https://doi.org/10.5281/zenodo.23732760>.

Análise original, figuras e tradução de Aridio Silva (@aridiosilva). A proposta analisada pertence a seu autor, R. Bhardwaj [9]; as figuras a redesenham e ampliam com atribuição.

Sobre o autor

Aridio Silva é pesquisador independente no Brasil e trabalha com arquitetura, segurança, governança e confiabilidade de sistemas de inteligência artificial autônomos e distribuídos.

Sua pesquisa se concentra em **Agentic AI, sistemas multiagentes, Edge AI, segurança de AI, Zero Trust, Security-by-Design, governança de AI, Spec-Driven Development** e **assurance** contínua de segurança.

É criador e pesquisador responsável pelo **SGAEIA — Secure Governed Autonomous Edge Intelligence Architecture**, uma iniciativa aberta de pesquisa sobre fundamentos arquiteturais para sistemas de AI autônomos seguros, governados, auditáveis e confiáveis em ambientes distribuídos edge-cloud.

Perfis de pesquisa

Aridio Silva — Pesquisador Independente, Brasil

- **ORCID:** <https://orcid.org/0009-0008-2411-6995>
- **Google Scholar:** <https://scholar.google.com/citations?user=rPn5O48AAAAJ>
- **Zenodo — Comunidade SGAEIA:** <https://zenodo.org/communities/sgaeia>
- **OpenAIRE:** <https://explore.openaire.eu/search/find?fv0=Aridio%20Silva&f0=q>
- **Medium:** <https://medium.com/@aridiosilva>
- **GitHub:** <https://github.com/aridiosilva>
- **LinkedIn:** <https://www.linkedin.com/in/aridio-silva-74997111/>
- **Repositório SGAEIA:** <https://github.com/aridiosilva/SGAEIA>
- **Artefato de pesquisa SGAEIA / DOI:** <https://doi.org/10.5281/zenodo.22557796>
- **Homepage:** <https://aridiosilva.com>
- **Homepage SGAEIA:** <https://aridiosilva.com/sgaeia>
- **SGAEIA no LinkedIn:** <https://www.linkedin.com/company/sgaeia/>
- **Kaggle:** <https://www.kaggle.com/aridiosilva2>
- **DEV Community:** <https://dev.to/aridiosilva>
- **Academia.edu:** <https://independent.academia.edu/AridioSilva>
- **Artigo 23 — Zenodo / DOI:** <https://doi.org/10.5281/zenodo.23732760>
- **Artigo 23 — Academia.edu:** [How to Reduce the Cost of AI Systems with LLMs Without Sacrificing Quality](#)
- **Artigo 23 — DEV Community:** [How to Reduce the Cost of AI Systems with LLMs Without Sacrificing Quality](#)
- **Artigo 23 — Medium:** [How to Reduce the Cost of AI Systems with LLMs Without Sacrificing Quality](#)

Figuras

A capa não é numerada. As Figuras 1–13 são numeradas em sequência, referenciadas no texto e fornecidas em PNG e SVG. Os ativos conceituais permanecem no limite de divulgação pública e estão licenciados sob CC BY 4.0.

Licença

Salvo indicação em contrário, o texto e as ilustrações conceituais originais deste artigo estão licenciados sob a **Creative Commons Attribution 4.0 International (CC BY 4.0)**.

© Aridio Silva @aridiosilva · Outubro 2026. O software e o artefato de pesquisa SGAEIA permanecem sujeitos à sua própria licença Apache 2.0.

Autonomous AI. Governed by Design. Trusted by Evidence.

Continuidade da série

Este trabalho integra a **SGAEIA Research Series — Article 23**. O catálogo público da série está disponível em <https://aridiosilva.com> e a presença institucional do projeto em <https://aridiosilva.com/sgaeia>. Registro persistente do artigo: <https://doi.org/10.5281/zenodo.23732760>.