

RELATÓRIO TÉCNICO · INFOGRÁFICO

Reduzir 70% do custo com LLMs sem perder qualidade

Revisão baseada em evidências de uma arquitetura de referência com seis alavancas

Aridio Silva · Pesquisador Independente · ORCID 0009-0008-2411-5995

Análise da proposta de R. Bhardwaj (LinkedIn, 26 out. 2026) · 55 fontes auditadas · 13 figuras



Roteiro do relatório

1 Introdução

2 Método

3 A tese central

4 Modelo de custo

5 As seis alavancas

6 Arquitetura de referência

7 Motor de contexto

8 Cache: prompt, semântico e KV

9 Roteamento e cascatas

10 Portões de qualidade

11 Decomposição e sensibilidade

12 Avaliação e KPIs

13 Observabilidade e governança

14 Roteiro de implementação

15 Ameaças à validade

16 Conclusão

A proposta se sustenta; os 70% pedem ressalvas

-70,7%

redução bruta do post: os cinco fatores se multiplicam

61-68%

economia líquida plausível, somando o custo extra das alavancas

±2,4 p.p.

ruído amostral numa avaliação com 500 itens: maior que as melhorias relatadas

- Todas as seis alavancas têm apoio em fontes primárias, e vários estudos relatam efeitos maiores que os do post.
- O valor de 70% é ilustrativo: omite roteador, juízes, embeddings e avaliação, e não mede a qualidade com intervalo de confiança.
- Métrica norte recomendada: custo por tarefa bem-sucedida (cost-of-pass).

O objeto: um infográfico com seis partes

“

Cortar custos com LLMs em até 70%, sem comprometer a inteligência, com roteamento de modelos, contexto, cache e fluxos agênticos mais inteligentes.

R. Bhardwaj, LinkedIn, 26 out. 2026 (paráfrase do post)

1

Manchete

US\$ 10.000 → US\$ 2.926 por mês

2

Seis alavancas

roteamento, contexto, prompt caching, cache semântico, loop agêntico, lote

3

Arquitetura

seis etapas + camada de governança

4

Cascata de custos

cinco reduções sequenciais, rotulada como ilustrativa

5

Tabela de qualidade

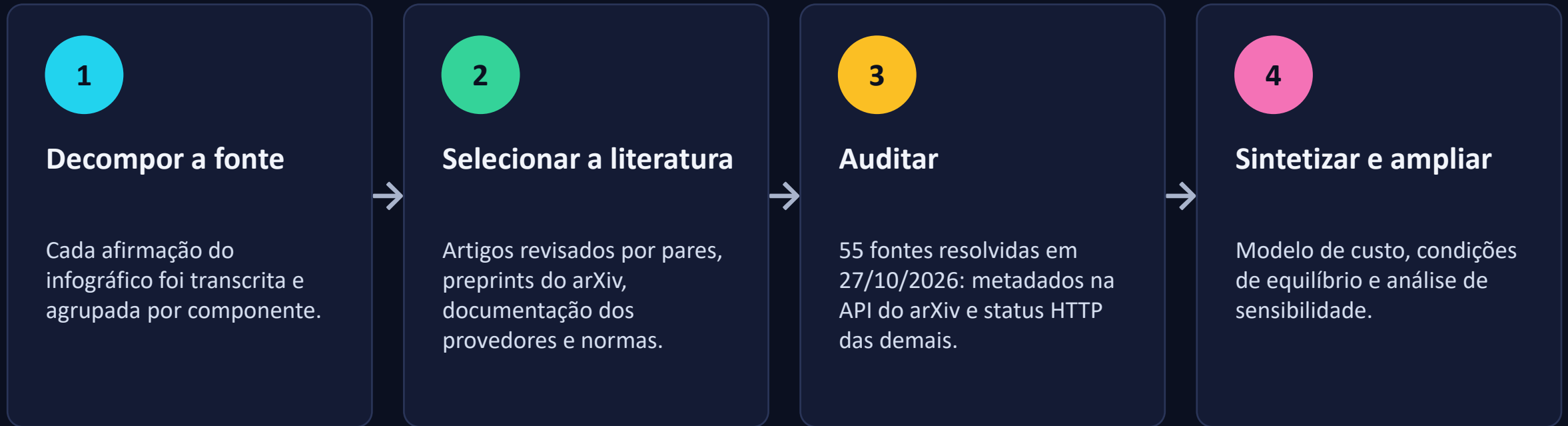
métricas antes e depois

6

Mensagem central

gaste inteligência só quando a tarefa precisar

Quatro passos, com auditoria de cada fonte



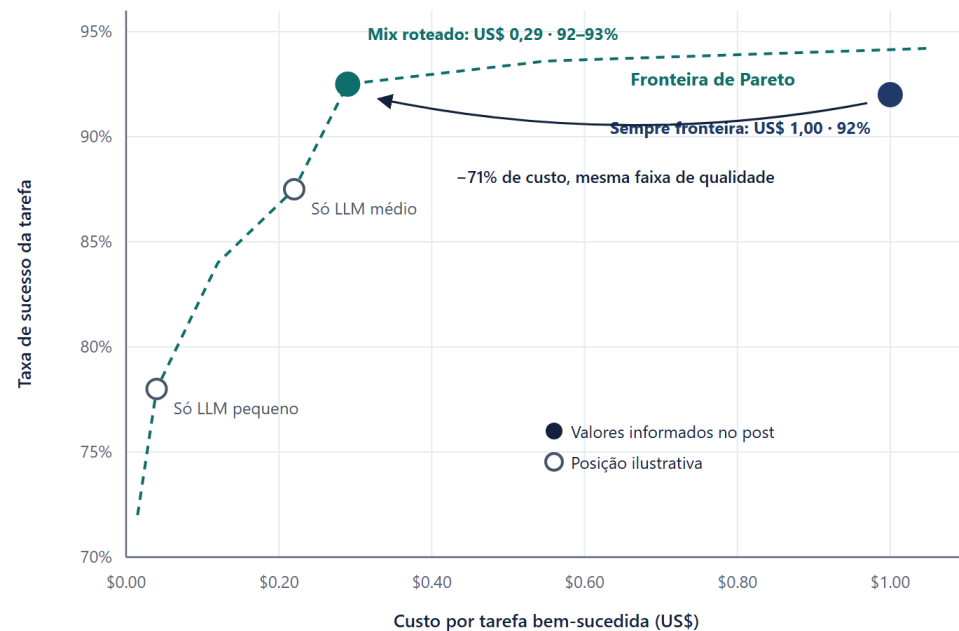
Limitações: revisão estruturada, não sistemática; nenhum experimento foi executado; preços dos provedores citados como lidos na data da auditoria.

Comprar qualidade onde ela é necessária

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.

Como ler

- A maioria das requisições reais é fácil: pagar preço de fronteira por elas compra capacidade ociosa.
- “Sempre fronteira” é um ponto dominado: o mix roteado atinge a mesma faixa de sucesso com 29% do custo por sucesso.
- Primeiro fixe um piso de qualidade (SLO); depois minimize o custo.
- Base: Chen 2023; Kapoor 2024; Liang 2022.

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.

© 2026 Aridio Silva (@aridiosilva) · Pesquisador Independente · ORCID 0009-0008-2411-5995 · Figuras sob licença CC BY 4.0

Custo por tarefa bem-sucedida

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

c = custo por tentativa · p = probabilidade de sucesso.
Formalizado como cost-of-pass (Erol et al. 2025).

Exemplo

Modelo A: US\$ 1,00 por tentativa, 92% de sucesso → US\$ 1,09 por sucesso

Modelo B: US\$ 0,10 por tentativa, 46% de sucesso → US\$ 0,22 por sucesso

10× mais barato por tentativa, só 5× por sucesso, antes de tratar as falhas.

Pontos fortes

- Coincide com os resultados de roteadores e cascatas.
- Uma única métrica legível pelo negócio.
- Vale para agentes e para chat.

Limites

- Depende da carga: se tudo é difícil, há pouco a rotear.
- “Mesma qualidade” precisa de prova estatística.
- A cauda longa crítica pode piorar sem afetar a média.

Cada alavanca ataca um fator da conta

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

Não é alavanca

Requisições por mês

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

1 - h

A4 Cache semântico

Taxa de falha do cache semântico

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

k

A5 Loop agêntico

Chamadas de LLM por requisição

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

m → p(m)

A1 Roteamento

Modelo escolhido → preço do token

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

T_u

A2 Contexto

Tokens de entrada sem cache

Trechos recuperados, histórico e instruções cobrados a preço cheio.

δ · T_c

A3 Prompt cache

Tokens de prefixo em cache

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

T_{out}

A2 Contexto

Tokens de saída

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

1 - β · b

A6 Lote

Desconto de lote na parcela em lote

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

C_{ovh}

Custo novo

Custo extra que as alavancas trazem

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

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

© 2026 Aridio Silva (@aridiosilva) · Pesquisador Independente · ORCID 0009-0008-2411-5995 · Figuras sob licença CC BY 4.0

Como ler

- N requisições × (1 - h) cache semântico × k chamadas × preço do modelo × tokens × desconto de lote, + overhead.
- Cada cartão colorido é uma alavanca (A1–A6).
- O cartão vermelho é o custo novo que as alavancas trazem, ausente no post.
- Preços de cache e lote: documentação Anthropic, OpenAI e Google (out. 2026).

As economias se multiplicam, não se somam

$$\begin{array}{c} 0,85 \\ \text{dedup} \end{array} \times \begin{array}{c} 0,60 \\ \text{roteamento} \end{array} \times \begin{array}{c} 0,75 \\ \text{tokens} \end{array} \times \begin{array}{c} 0,85 \\ \text{caching} \end{array} \times \begin{array}{c} 0,90 \\ \text{lote} \end{array} = \begin{array}{c} 0,293 \\ -70,7\% \end{array}$$

Composição

15% + 40% + 25% + 15% + 10%
somariam 105%. Como cada corte age
sobre o que sobrou, o total é 70,7%.

Sobreposição

Depois que o roteamento barateia o
modelo, cortar tokens economiza ao
preço menor. A ordem muda a fatia de
cada etapa, não o produto.

Lote implica $b = 20\%$

Com desconto $\beta = 50\%$, o fator $0,90 = 1 - 0,5 \cdot b \Rightarrow$ uma requisição em cinco
pode esperar até 24 h.

Alvo, evidência, risco e trade-off

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).	méd-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).	médio	médio	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	médio	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-méd	médio	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).

© 2026 Aridio Silva (@aridiosilva) · Pesquisador Independente · ORCID 0009-0008-2411-5995 · Figuras sob licença CC BY 4.0

Leitura rápida

- Maior efeito: roteamento (até 98% no FrugalGPT).
- Menor risco: prompt caching e lote.
- Maior risco à qualidade: cache semântico (acertos falsos).
- Nos próximos slides: uma alavanca por vez.

Roteamento inteligente de modelos

A1

Envia cada requisição ao modelo mais barato capaz de respondê-la bem. A cascata é a variante sequencial: tenta o barato, verifica e só escala se falhar.

até 98%

de redução de custo com a
acurácia do GPT-4

FrugalGPT · Chen 2023

>2×

mais barato sem perder
qualidade

RouteLLM · Ong 2024

-40%

chamadas ao modelo
grande, mesma qualidade

Hybrid LLM · Ding 2024

~40%

do custo do GPT-4 em
raciocínio

MoT cascade · Yue 2023

Trade-off: roteadores precisam de dados e envelhecem com os modelos; errar o “fácil” prejudica justamente as consultas raras e críticas.

Engenharia de contexto

A2

Decidir quais tokens entram na janela de contexto, e em que ordem: o menor conjunto de tokens de alto sinal para a tarefa (Anthropic 2025).

meio

é onde os modelos usam pior a evidência

Lost in the middle · Liu 2024

até 20×

compressão de prompt com pouca perda

LLMLingua · Jiang 2023

↑ tokens

degradam o desempenho mesmo em tarefas simples

Context rot · Chroma 2025

Trade-off: cortar demais perde o trecho que importava (recall); comprimir acrescenta uma chamada e latência.

Prompt caching

A3

Quando muitas requisições repetem o mesmo prefixo (instruções, schemas, exemplos, documento), o provedor reaproveita o estado já processado e cobra menos.

0,1×

preço da leitura em cache (escrita 1,25×
em 5 min)

Anthropic, doc. 2026

até 95%

de desconto na entrada em cache; mín.
1.024 tokens

OpenAI, doc. 2026

8×

menos tempo até o 1º token em GPU

Prompt Cache · Gim 2024

Trade-off: o prefixo precisa ser idêntico byte a byte; caches compartilhados vazam tempo entre usuários (7 provedores auditados, Gu 2025).

Cache semântico



Guarda respostas inteiras e devolve uma quando a nova consulta tem significado próximo de outra já respondida, medido por similaridade de embeddings.

~31%

das consultas eram repetidas no cenário estudado

MeanCache · Gill 2024

1 busca

substitui recuperação, geração e checagens num acerto

GPTCache · Bang 2023

3 riscos

acerto falso, resposta velha, vazamento entre tenants

OWASP LLM Top 10 2025

Trade-off: é a única alavanca que devolve uma resposta errada sem nenhum modelo envolvido; por isso entra por último no roteiro.

Loop agêntico e processamento em lote

A5

Otimização do loop agêntico (k)

- Cada passo do agente é uma chamada paga; passos redundantes, retries e repasses multiplicam k .
- Baselines simples igualaram agentes de ponta por uma fração do custo (Kapoor 2024).
- Comece pelo padrão mais simples: uma chamada, depois um fluxo fixo (Anthropic 2024).
- Trade-off: limites rígidos podem parar o agente pouco antes do sucesso; defina-os pelos traces.

A6

Processamento em lote ($\beta \cdot b$)

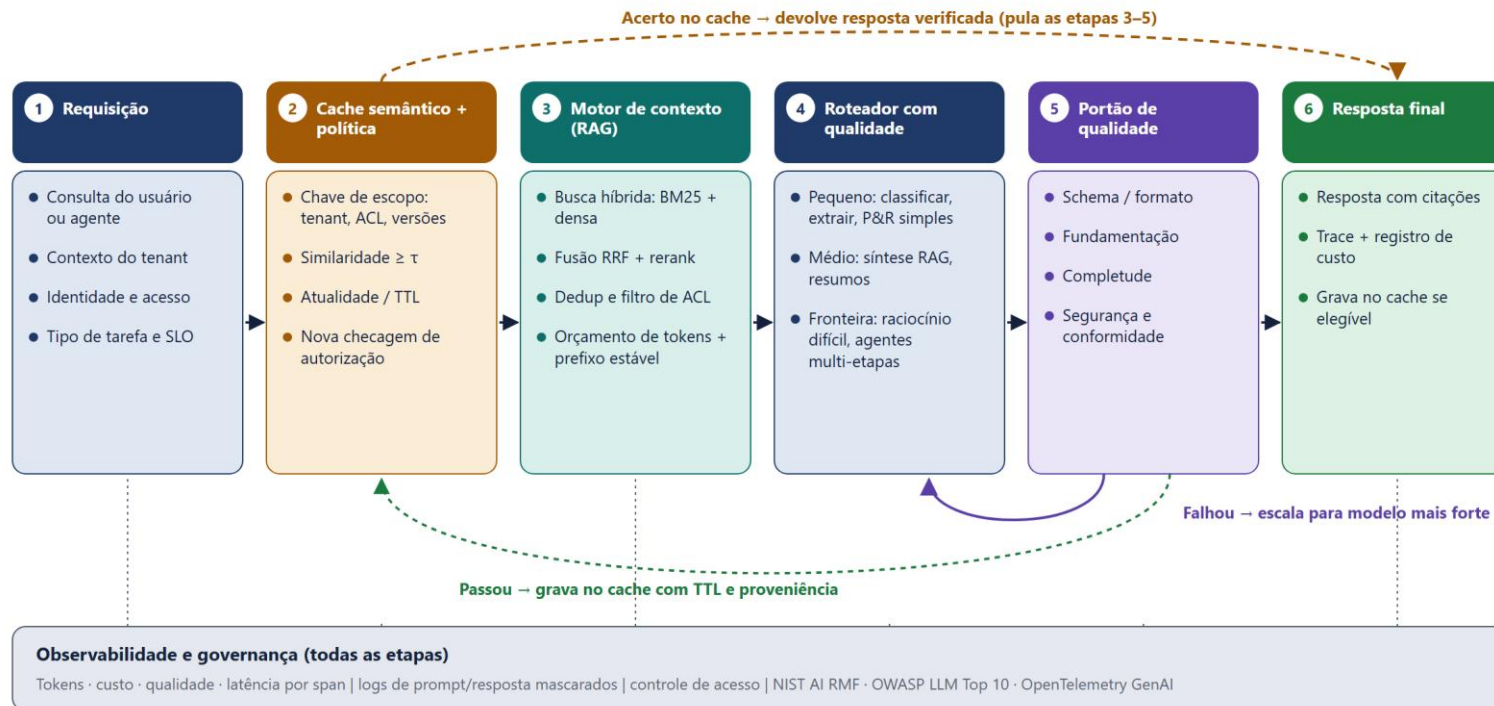
- Trabalho que pode esperar vai em bloco, de forma assíncrona: enriquecimento, backlogs, avaliações, relatórios.
- 50% de desconto e resultado em até 24 h na Anthropic, OpenAI e Google.
- Na Anthropic o desconto se acumula com o prompt caching.
- Trade-off: só serve para trabalho tolerante a latência; acrescenta fila.

Seis etapas e três laços de controle

FIGURA 4

Arquitetura de referência: stack GenAI orientada a qualidade e custo

Caminho da requisição e laços de controle



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.

© 2026 Aridio Silva (@aridiosilva) · Pesquisador Independente · ORCID 0009-0008-2411-5995 · Figuras sob licença CC BY 4.0

Os três laços

- **Âmbar:** acerto no cache devolve a resposta verificada e pula as etapas 3–5.
- **Violeta:** se o portão reprova, escala para um modelo mais forte.
- **Verde:** só o que passou no portão é gravado no cache, com TTL e proveniência.
- **Base:** Lewis 2020 (RAG); NIST AI RMF; OWASP 2025.

Três princípios de projeto



1

Decisões baratas primeiro

Busca no cache e checagens determinísticas custam muito menos que uma chamada de modelo, então rodam antes dela.

2

Fechar o laço

Só respostas aprovadas no portão entram no cache, o que impede que erros sejam repetidos.

3

Recuperação autocorretiva

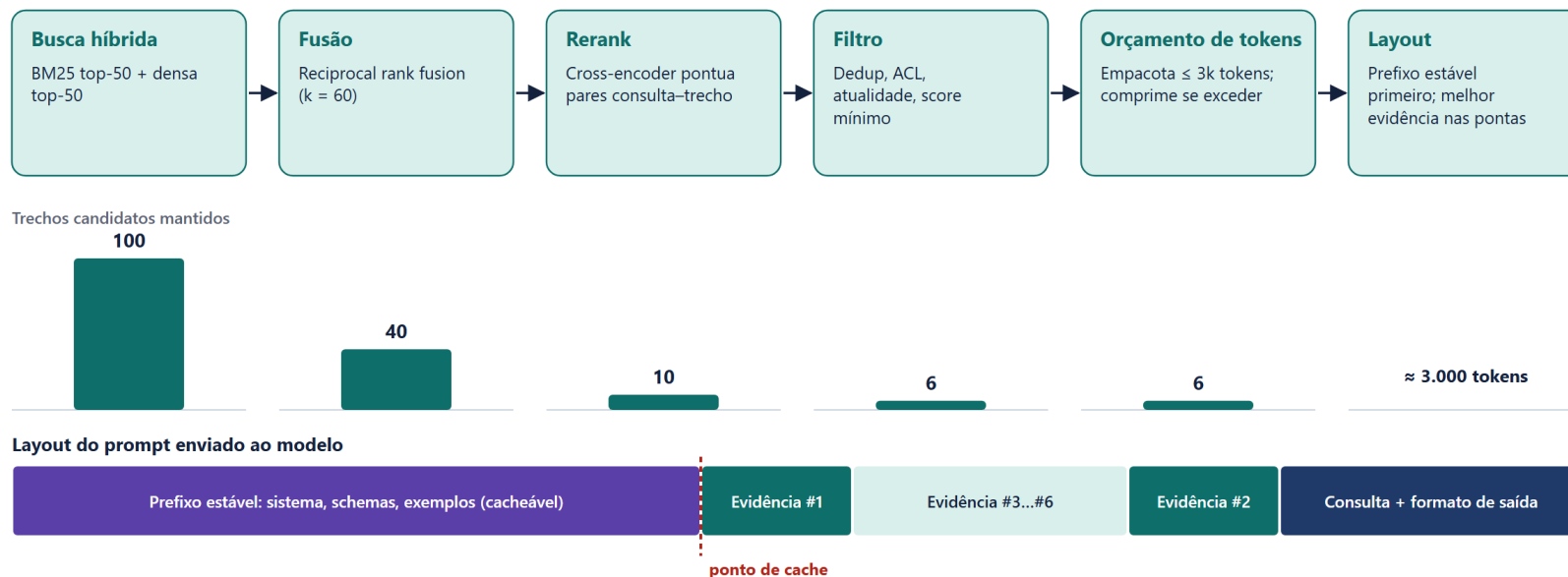
Self-RAG (Asai 2023) e Corrective RAG (Yan 2024) checam a recuperação dentro do laço de geração.

De 100 candidatos a um prompt com orçamento

FIGURA 5

Motor de otimização de contexto: de 100 candidatos a um prompt com orçamento

Contagens ilustrativas



A evidência mais forte vai no início e no fim: os modelos usam pior a informação do meio (Liu et al. 2023). O prefixo cacheável deve ser idêntico byte a byte entre requisições.

Fontes: Robertson & Zaragoza 2009 (BM25); Cormack et al. 2009 (RRF); Nogueira & Cho 2019 (rerank); Liu et al. 2023; Jiang et al. 2023.

© 2026 Aridio Silva (@aridiosilva) · Pesquisador Independente · ORCID 0009-0008-2411-5995 · Figuras sob licença CC BY 4.0

Etapas

- BM25 acha termos exatos; a busca densa acha paráfrases.
- RRF funde as listas: $\sum 1/(60 + \text{posição})$, sem calibrar scores.
- Cross-encoder reordena; filtros tiram duplicatas, ACL e versões velhas.
- Prefixo estável primeiro (cacheável); melhor evidência nas pontas.

Três camadas que costumam ser confundidas

FIGURA 6
Três camadas de cache frequentemente confundidas

O que cada camada reutiliza e arrisca

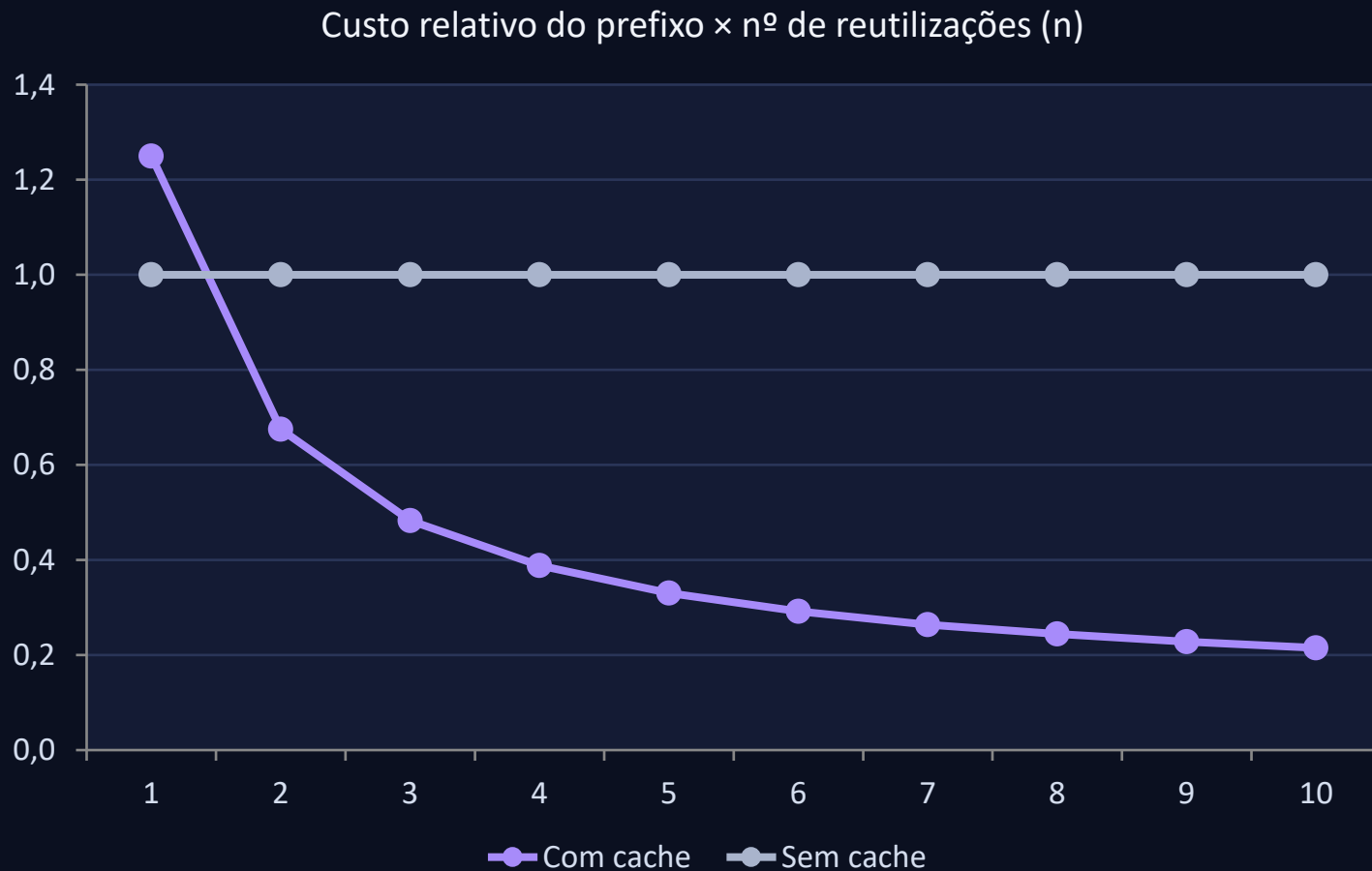
Camada	Reutiliza	Regra de match	Economiza	Principal risco
Cache semântico Camada de aplicação (A4)	A resposta final inteira	Similaridade de embedding $\geq \tau$ dentro do escopo tenant/ACL/versão	O pipeline inteiro: custo e latência	Acertos falsos, respostas velhas, vazamento entre tenants
Prompt / 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 $0,1\times$ 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. 2024; Gu et al. 2025; docs Anthropic e OpenAI (2026).
© 2026 Aridio Silva (@aridiosilva) · Pesquisador Independente · ORCID 0009-0008-2411-5995 · Figuras sob licença CC BY 4.0

A diferença

- Semântico: reaproveita a resposta inteira; é o maior ganho e o maior risco.
- Prompt cache: reaproveita o prefixo processado; é barato e seguro se não for compartilhado.
- KV cache: assunto de quem hospeda o modelo (PagedAttention, RadixAttention).

Quando o prompt caching se paga



custo com cache / sem cache

$$[1,25 + 0,1 \cdot (n - 1)] / n$$

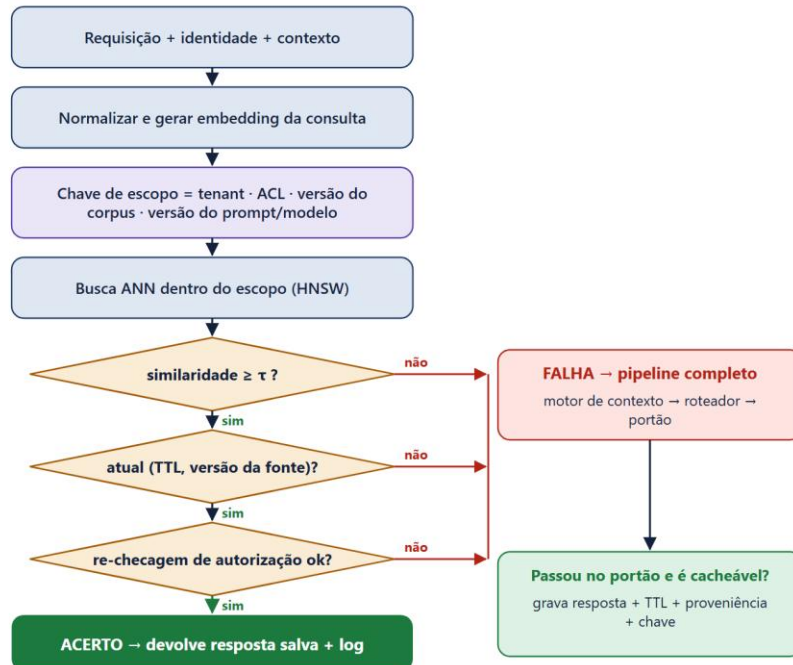
- Paga a partir do 2º uso: 1,35 contra 2,0.
- Com $n = 10$ o prefixo custa 21,5% do preço cheio.
- Só dá prejuízo se o prefixo gravado nunca for reutilizado antes de expirar.
- Multiplicadores da Anthropic: escrita 1,25×, leitura 0,1×.

Similar, atual e autorizado

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.

© 2026 Aridio Silva (@aridiosilva) · Pesquisador Independente · ORCID 0009-0008-2411-5995 · Figuras sob licença CC BY 4.0

Quatro controles

- Chave de escopo: tenant, ACL, versão do corpus e do prompt.
- Limiar τ calibrado por domínio, com limite de acerto falso como SLO.
- Atualidade: TTL e invalidação quando a fonte muda.
- Gravar só depois do portão, com proveniência.

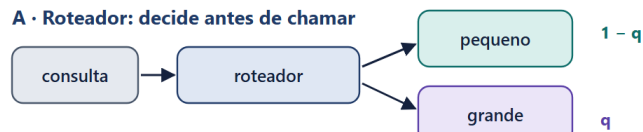
Quando a cascata deixa de compensar

FIGURA 8

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

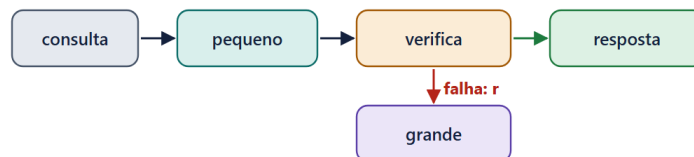
Custo esperado relativo a usar sempre o modelo grande

A · Roteador: decide antes de chamar



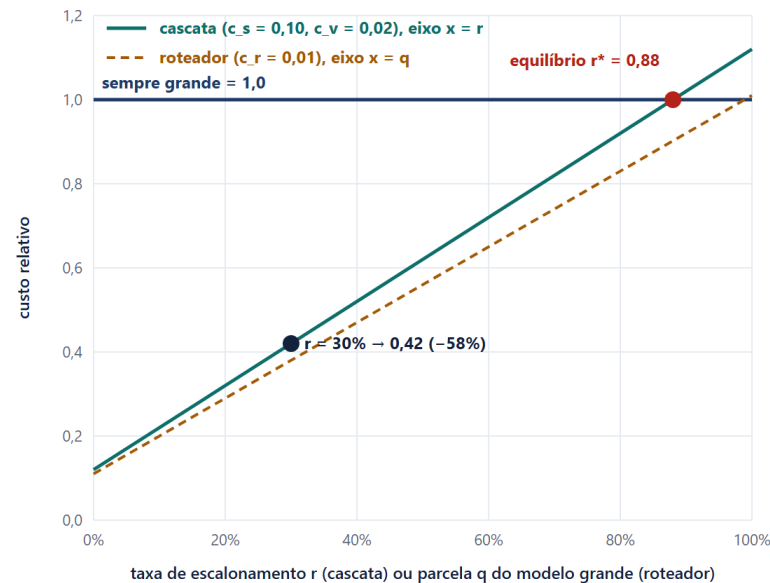
$$E[C_{\text{router}}] = c_r + (1 - q) \cdot c_s + q \cdot c_L$$

B · Cascata: tenta o pequeno, verifica, escala



$$E[C_{\text{cascade}}] = c_s + c_v + r \cdot c_L$$

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



A conta

- Cascata: $E[C] = c_s + c_v + r \cdot c_L$.
- Só compensa se $r < 1 - (c_s + c_v)/c_L$.
- Com $c_s = 0,10$ e $c_v = 0,02$: equilíbrio em $r^* = 0,88$.
- Com $r = 30\%$: custo 0,42, ou -58%.

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.

© 2026 Aridio Silva (@aridiosilva) · Pesquisador Independente · ORCID 0009-0008-2411-5995 · Figuras sob licença CC BY 4.0

Roteador ou cascata?

	Roteador (decide antes)	Cascata (tenta e escala)
Latência	Uma chamada de modelo + um roteador pequeno	Escaladas pagam duas chamadas em série
Qualidade do sinal	Prevê a dificuldade antes de ver a resposta	Julga uma resposta real: mais confiável
Dados necessários	Dados rotulados ou de preferência	Um verificador ou checagem de consistência
Melhor para	Tráfego interativo, sensível à latência	Tráfego em lote ou sensível à acurácia

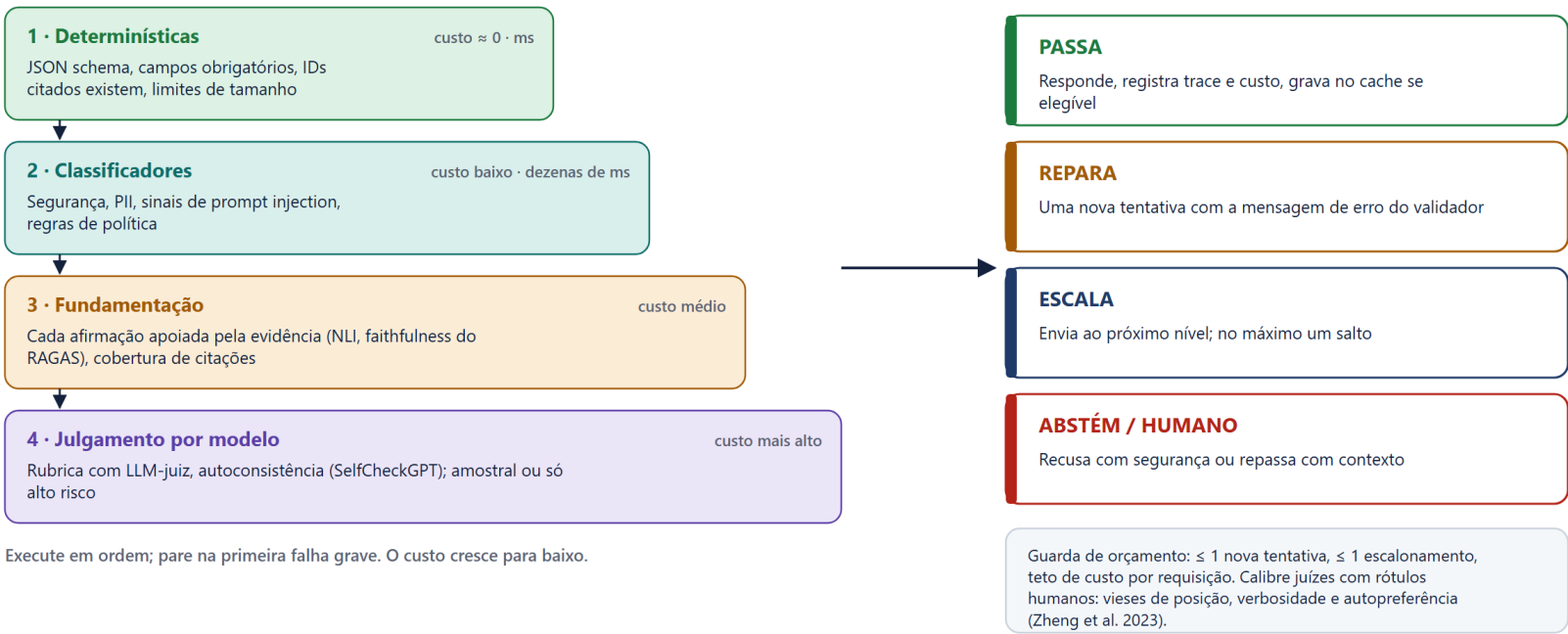
Sinais usados na literatura: dificuldade prevista (Ding 2024), preferência humana (Ong 2024), consistência de respostas (Yue 2023), autoverificação com POMDP (Aggarwal 2023), incerteza por token (Gupta 2024).

Checagens baratas primeiro, escalonamento limitado

FIGURA 9

Portão de qualidade: checagens mais baratas primeiro, escalonamento limitado

Camadas de checagem e desfechos



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.

© 2026 Aridio Silva (@aridiosilva) · Pesquisador Independente · ORCID 0009-0008-2411-5995 · Figuras sob licença CC BY 4.0

Por que funciona

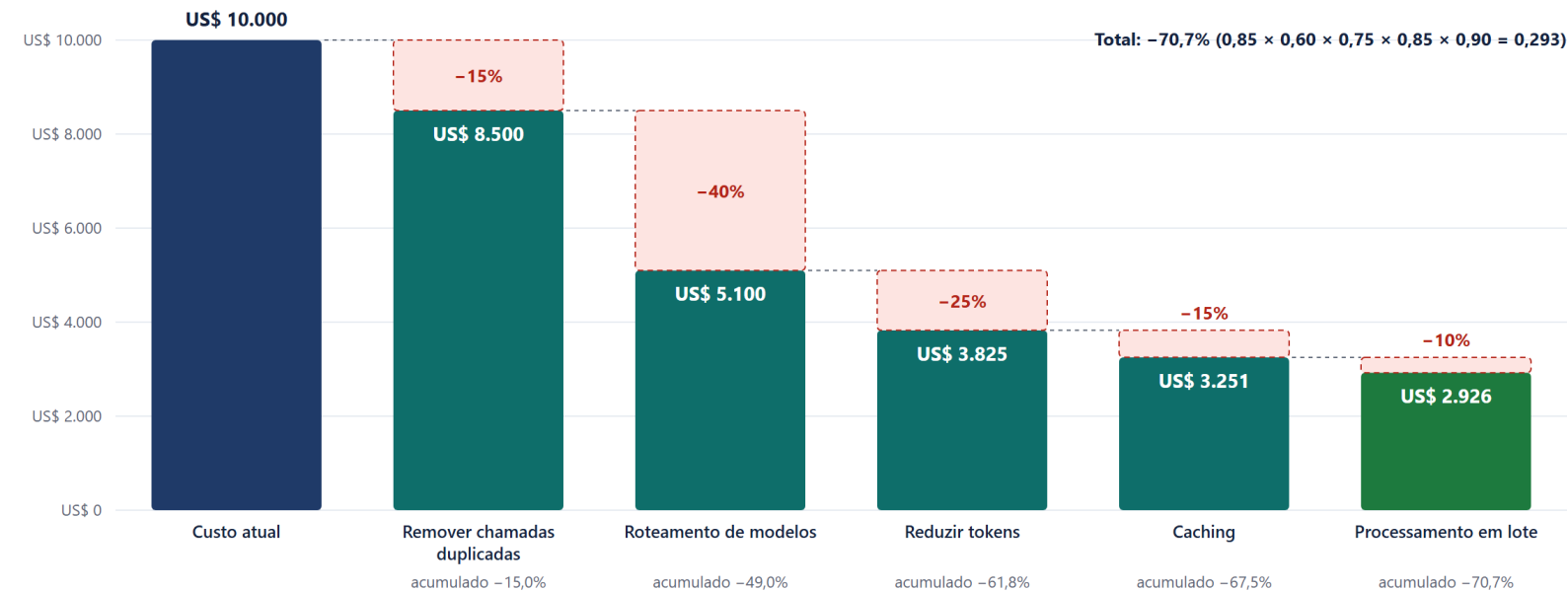
- É o portão que torna seguro o roteamento agressivo.
- Determinísticas e classificadores pegam a maioria das falhas a custo quase zero.
- LLM-juiz concorda >80% com humanos, mas tem vieses de posição, verbosidade e autopreferência (Zheng 2023).
- Guarda: ≤ 1 retry, ≤ 1 escalonamento, teto de custo (OWASP LLM10).

A cascata do post, redesenhada em escala

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.

© 2026 Aridio Silva (@aridiosilva) · Pesquisador Independente · ORCID 0009-0008-2411-5995 · Figuras sob licença CC BY 4.0

Conferência

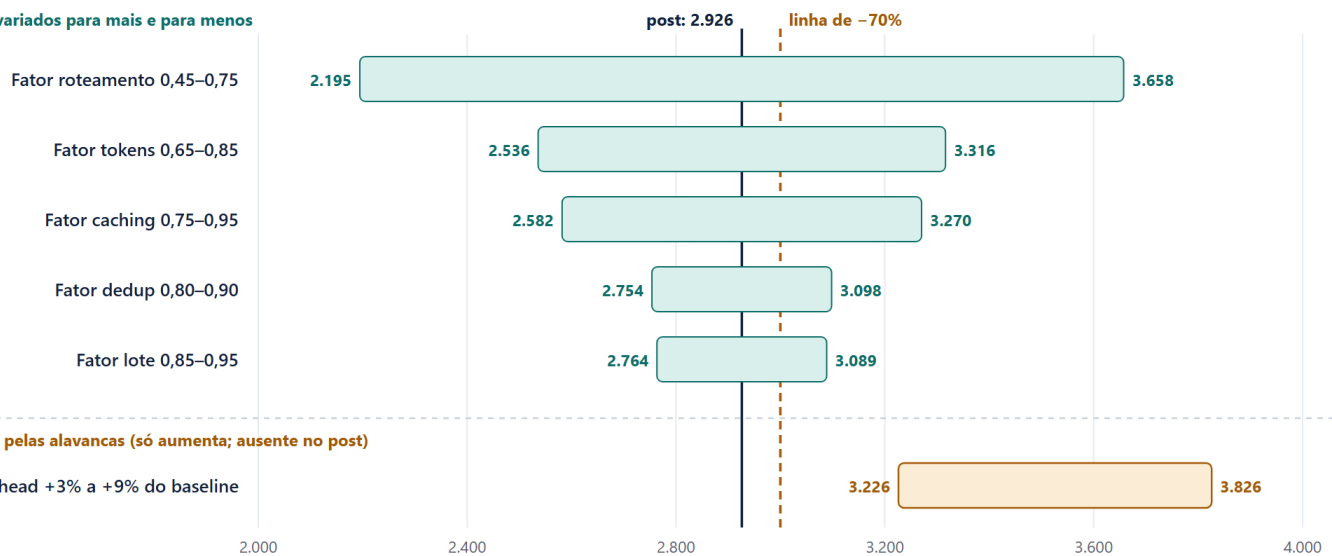
- $10.000 \times 0,85 \times 0,60 \times 0,75 \times 0,85 \times 0,90 = 2.926$: a conta fecha.
- O roteamento é a maior etapa: 34 dos 70,7 pontos.
- O post rotula o gráfico como ilustrativo: são metas, não medições.

Quão robusta é a afirmação de 70%?

FIGURA 11
Sensibilidade: quão robusta é a afirmação de 70%?

Custo mensal final, variando uma premissa por vez

A • Fatores do post, variados para mais e para menos



B • Custo adicionado pelas alavancas (só aumenta; ausente no post)

Overhead +3% a +9% do baseline

Só o roteamento move o resultado de -64% para -78%. Somando um overhead realista, a economia líquida fica em cerca de 61–68%.

Modelo e faixas do autor. Linhas A: sensibilidade uma-a-uma em torno dos fatores do post. Linha B: overhead somado. Ilustrativo.

© 2026 Aridio Silva (@aridiosilva) • Pesquisador Independente • ORCID 0009-0008-2411-5995 • Figuras sob licença CC BY 4.0

Resultado

- Só o roteamento move o total de -64% a -78%.
- Lote e deduplicação são os que menos pesam.
- A linha âmbar (B) não é uma variação: é o custo que as alavancas somam, por isso só aumenta.
- Com overhead de 3–9% do baseline, a economia líquida fica em ~61–68%.

O que as alavancas acrescentam à conta

Item	Introduzido por	Como manter pequeno
Embeddings, 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	Determinísticas primeiro; juízes por amostragem
Escritas no cache (1,25×–2×)	A3	Cachear só prefixos que serão reutilizados
Rodadas de avaliação	Todas	Rodar avaliações pela API de lote (–50%)
Engenharia e operação	Todas	Fasear; ganhos rápidos primeiro

Com 3–9% do baseline em overhead, a economia líquida cai de 70,7% para cerca de 61–68%.

“Mesma qualidade” precisa de prova estatística

$\pm 2,4$ p.p.

intervalo de 95% para 92% de sucesso com 500 itens

$$EP = \sqrt{(0,92 \cdot 0,08 / 500)} \approx 1,2 \text{ p.p.}$$

O post reporta 92% → 92–93%, 90% → 90–91% e 0,5% → 0,4%: tudo dentro do ruído (Miller 2024).

1

Desenho pareado

Avalie os dois sistemas nos mesmos itens.

2

Margem de não inferioridade

Pré-registre, por exemplo: “sucesso no máximo 1 ponto menor”.

3

Tamanho da amostra

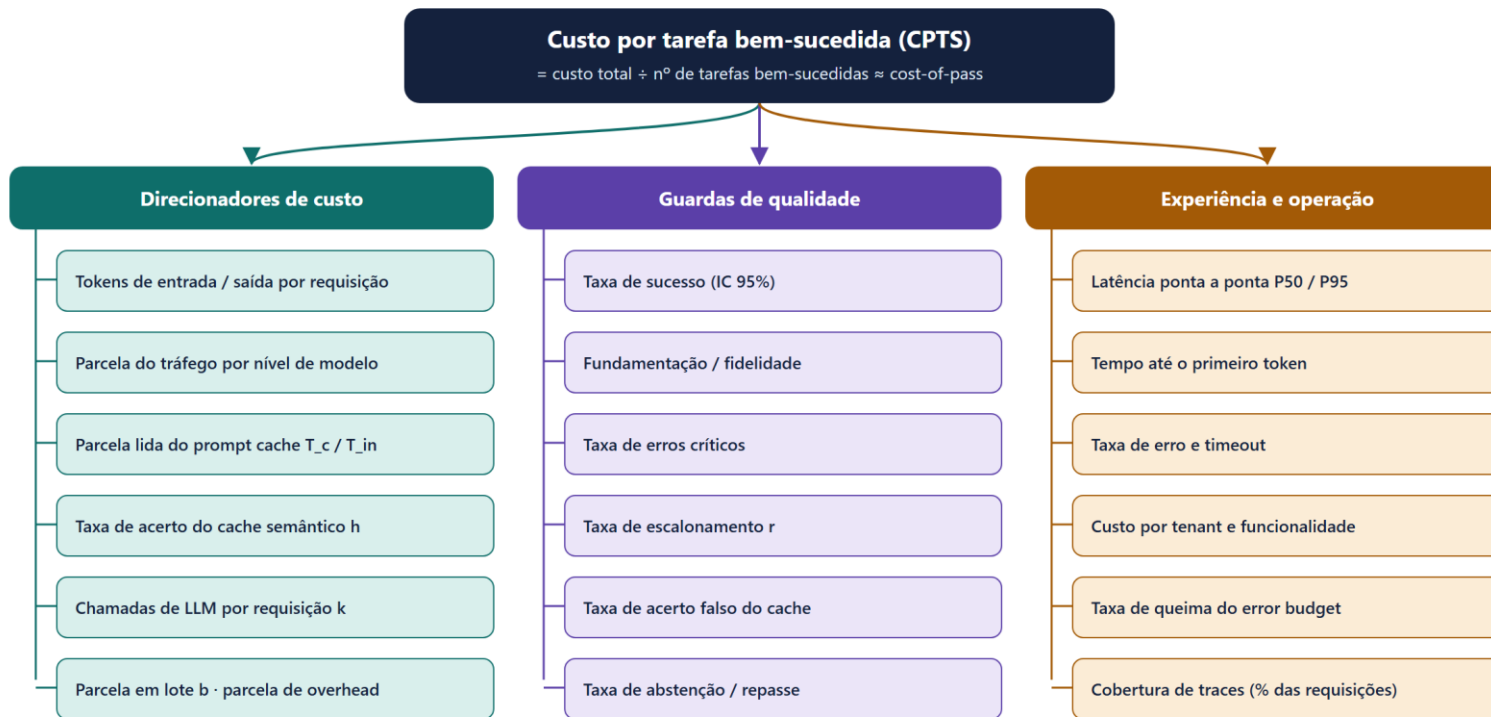
Dimensione o conjunto para que o intervalo fique mais estreito que a margem.

Uma métrica norte, três famílias

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.

© 2026 Aridio Silva (@aridiosilva) · Pesquisador Independente · ORCID 0009-0008-2411-5995 · Figuras sob licença CC BY 4.0

O que medir

- Norte: custo por tarefa bem-sucedida (Erol 2025).
- Custo: tokens/req, mix de modelos, leitura do cache, taxa de acerto, k , lote.
- Qualidade: sucesso com IC 95%, fundamentação, erros críticos, escalonamento, acerto falso.
- Operação: P50/P95, TTFT, timeouts, custo por tenant, cobertura de traces.

Sem telemetria por requisição, nada se ajusta

1

OpenTelemetry GenAI

Atributos padronizados de modelo, tokens e operações por span: custo por etapa portátil entre fornecedores.

2

NIST AI RMF + AI 600-1

Funções Govern, Map, Measure e Manage, e o Perfil de IA Generativa.

3

OWASP LLM Top 10 2025

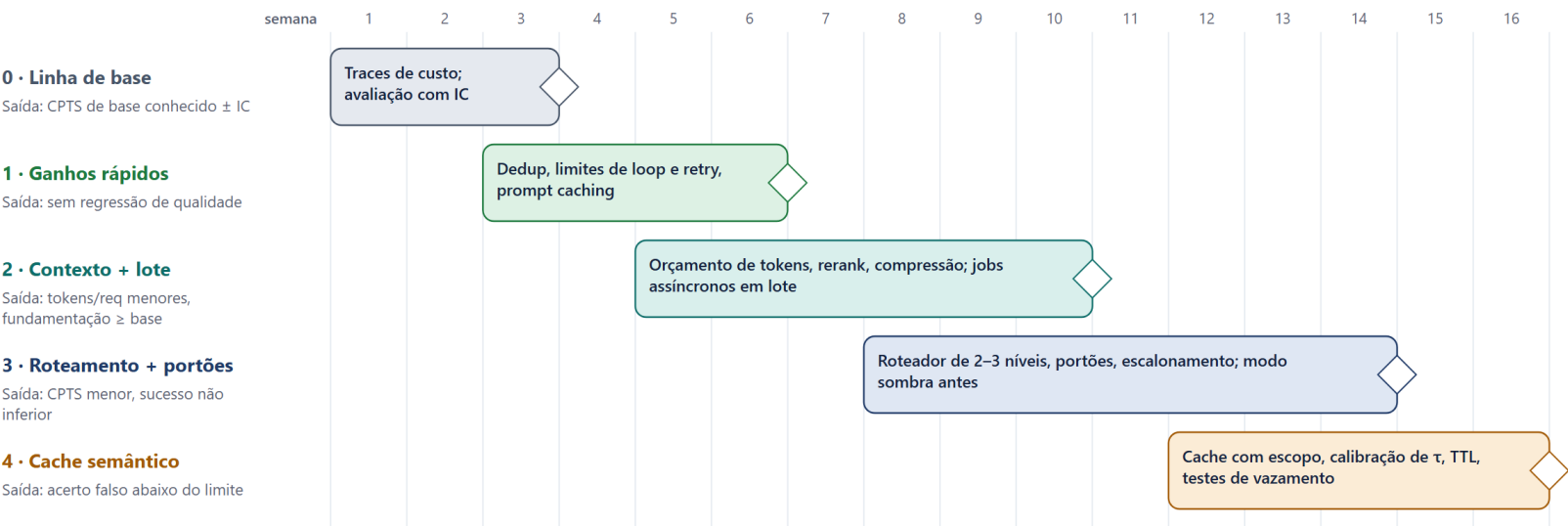
Prompt injection, vazamento de informação sensível, fraquezas de vetores e embeddings, consumo ilimitado.

Dívida técnica oculta (Sculley 2015): cada cache, roteador e juiz é mais um componente que pode se desviar. Cada um precisa de um responsável, uma métrica e um alerta.

Baixo risco primeiro, cada fase com portão

FIGURA 13

Roteiro de implementação: alavancas de baixo risco primeiro, cada fase com portão



◇ = portão de fase: só avance quando o critério de saída se confirmar no conjunto de avaliação.

Roteiro do autor; a sequência segue Sculley et al. 2015, Kapoor et al. 2024, Miller 2024 e NIST AI 600-1.

© 2026 Aridio Silva (@aridiosilva) · Pesquisador Independente · ORCID 0009-0008-2411-5995 · Figuras sob licença CC BY 4.0

A ordem importa

- 0 · Medir: traces de custo e avaliação com IC.
- 1 · Ganhos rápidos: dedup, limites de loop, prompt caching.
- 2 · Contexto e lote.
- 3 · Roteamento e portões, em modo sombra antes.
- 4 · Cache semântico por último: exige a governança mais madura.

Onde os números podem não valer



Dependência da carga

Tarefas difíceis, únicas e longas têm pouco a rotear ou cachear.



Volatilidade de preços

Multiplicadores mudam várias vezes por ano; o tamanho da economia muda junto.



Latência

P95 menor é plausível, mas cascatas somam chamadas em série. Meça por rota.



Validade da avaliação

Roteadores ajustados no próprio conjunto de teste superestimam ganhos.



Segurança dos caches

Caches compartilhados criam canais de informação entre usuários.



Escopo

Análise baseada na literatura; as Figuras 8 e 11 são modelos ilustrativos.

Gaste inteligência só onde a tarefa precisa, e meça por sucesso

Sólida

As seis alavancas têm apoio em fontes primárias, algumas com efeitos maiores que os do post.

Realista

Economia líquida plausível de 60–70% em cargas com muitas requisições simples ou repetidas.

Segura

Medir primeiro, colher o baixo risco e só então rotear e cachear atrás de portões e testes de não inferioridade.

Como citar: Silva, Aridio. (2026). Reduzir em 70% o custo com LLMs sem perder qualidade: uma revisão baseada em evidências de uma arquitetura de referência com seis alavancas (Versão 1.0) [Relatório técnico]. ORCID 0009-0008-2411-5995.



Conceitos-chave

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.

Conceitos-chave

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.

Conceitos-chave

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.

Conceitos-chave

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 em 27/10/2026

- [1] Aggarwal, P., Madaan, A., Anand, A., Potharaju, S. P., et al. (2023). *AutoMix: Automatically mixing language models*. arXiv preprint arXiv:2310.12963. arxiv.org/abs/2310.12963
- [2] Anthropic (2024, dez. 19). *Building effective AI agents*. Anthropic Engineering. www.anthropic.com/research/building-effective-agents
- [3] Anthropic (2025, set. 29). *Effective context engineering for AI agents*. Anthropic Engineering. www.anthropic.com/research/effective-context-engineering-for-ai-agents
- [4] Anthropic (2026). *Batch processing*. Claude Developer Platform documentation. platform.claude.com/docs/en/build-with-claude/batch-processing
- [5] Anthropic (2026). *Prompt caching*. Claude Developer Platform documentation. platform.claude.com/docs/en/build-with-claude/prompt-caching
- [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. arxiv.org/abs/2310.11511
- [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. aclanthology.org/2023.nlposs-1.24/
- [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. sre.google/sre-book/service-level-objectives/
- [9] Bhardwaj, R. (2026, out. 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. www.linkedin.com/posts/rajeev-bhardwaj_genai-agentical-llmoptimization-share-7520411441674387456-qL-Q
- [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. arxiv.org/abs/2305.05176
- [11] Chroma Research (2025). *Context rot: How increasing input tokens impacts LLM performance*. Chroma technical report. www.trychroma.com/research/context-rot
- [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. doi.org/10.1145/1571941.1572114
- [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. arxiv.org/abs/2404.14618
- [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. arxiv.org/abs/2504.13359

Referências auditadas em 27/10/2026

[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. arxiv.org/abs/2309.15217

[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. arxiv.org/abs/2403.02694

[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. arxiv.org/abs/2311.04934

[18] Google (2026). *Batch API*. Gemini API documentation. ai.google.dev/gemini-api/docs/batch-api

[19] Google (2026). *Context caching*. Gemini API documentation. ai.google.dev/gemini-api/docs/caching

[20] Gu, C., Li, X. L., Kuditipudi, R., Liang, P., & Hashimoto, T. (2025). *Auditing prompt caching in language model APIs*. arXiv preprint arXiv:2502.07776. arxiv.org/abs/2502.07776

[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. arxiv.org/abs/2404.10136

[22] Han, T., Wang, Z., Fang, C., Zhao, S., et al. (2024). *Token-budget-aware LLM reasoning*. arXiv preprint arXiv:2412.18547. arxiv.org/abs/2412.18547

[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. arxiv.org/abs/2310.05736

[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. arxiv.org/abs/2310.06839

[25] Kapoor, S., Stroebl, B., Siegel, Z. S., Nadgir, N., & Narayanan, A. (2024). *AI agents that matter*. arXiv preprint arXiv:2407.01502. arxiv.org/abs/2407.01502

[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. arxiv.org/abs/2309.06180

[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. arxiv.org/abs/2005.11401

[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. arxiv.org/abs/2211.09110

Referências auditadas em 27/10/2026

[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/tacl_a_00638. arxiv.org/abs/2307.03172

[30] LMSYS (2024–2026). *RouteLLM: A framework for serving and evaluating LLM routers [Software]*. GitHub. github.com/lm-sys/RouteLLM

[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. arxiv.org/abs/1603.09320

[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. arxiv.org/abs/2303.08896

[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.

arxiv.org/abs/2312.15234

[34] Microsoft (2023–2026). *LLMLingua [Software]*. GitHub.

github.com/microsoft/LLMLingua

[35] Miller, E. (2024). *Adding error bars to evals: A statistical approach to language model evaluations*. arXiv preprint arXiv:2411.00640. arxiv.org/abs/2411.00640

[36] National Institute of Standards and Technology (2023). *Artificial intelligence risk management framework (AI RMF 1.0)*. NIST AI 100-1. doi.org/10.6028/NIST.AI.100-1

[37] National Institute of Standards and Technology (2024). *Artificial intelligence risk management framework: Generative artificial intelligence profile*. NIST AI 600-1.

doi.org/10.6028/NIST.AI.600-1

[38] Nogueira, R., & Cho, K. (2019). *Passage re-ranking with BERT*. arXiv preprint arXiv:1901.04085. arxiv.org/abs/1901.04085

[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.

arxiv.org/abs/2406.18665

[40] OpenAI (2026). *Batch API*. OpenAI API documentation.

developers.openai.com/api/docs/guides/batch

[41] OpenAI (2026). *Prompt caching*. OpenAI API documentation.

developers.openai.com/api/docs/guides/prompt-caching

[42] OpenTelemetry (2026). *Semantic conventions for generative AI systems*. GitHub: open-telemetry/semantic-conventions-genai. github.com/open-telemetry/semantic-conventions-genai

Referências auditadas em 27/10/2026

[43] OWASP GenAI Security Project (2025). *OWASP Top 10 for LLM applications 2025*. OWASP Foundation. genai.owasp.org/llm-top-10/

[44] Ragas contributors (2023–2026). *Ragas: Evaluation toolkit for LLM applications [Software]*. GitHub. github.com/vibrantlabsai/ragas

[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>. aclanthology.org/D19-1410/

[46] Robertson, S., & Zaragoza, H. (2009). *The probabilistic relevance framework: BM25 and beyond*. Foundations and Trends in Information Retrieval, 3(4), 333–389. doi.org/10.1561/15000000019

[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. proceedings.neurips.cc/paper/2015/hash/86df7dcfd896fcdf2674f757a2463eba-Abstract.html

[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. arxiv.org/abs/2304.09542

[49] Yan, S.-Q., Gu, J.-C., Zhu, Y., & Ling, Z.-H. (2024). *Corrective retrieval augmented generation*. arXiv preprint arXiv:2401.15884. arxiv.org/abs/2401.15884

[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. arxiv.org/abs/2210.03629

[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. arxiv.org/abs/2310.03094

[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. arxiv.org/abs/2306.05685

[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. arxiv.org/abs/2312.07104

[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. arxiv.org/abs/2404.14294

[55] Zilliz (2023–2026). *GPTCache: Semantic cache for LLMs [Software]*. GitHub. github.com/zilliztech/GPTCache