doi.org/10.51891/rease.v12i7.28657

OTIMIZAÇÃO DE RECURSOS E RESILIÊNCIA COGNITIVA EM AGENTES AUTÔNOMOS VIA PARADIGMA ACTION-RAG

RESOURCE OPTIMIZATION AND COGNITIVE RESILIENCE IN AUTONOMOUS AGENTS VIA THE ACTION-RAG PARADIGM

OPTIMIZACIÓN DE RECURSOS Y RESILIENCIA COGNITIVA EN AGENTES AUTÓNOMOS VÍA PARADIGMA ACTION-RAG

Gabriel Lima Scheffler[1]
Agnaldo da Costa[2]
Arlete Teresinha Beuren[3]

RESUMO: A escalabilidade de modelos de inteligência artificial autônomos é restringida atualmente pela volatilidade das APIs e a saturação da janela de atenção do Transformer, no qual a injeção exaustiva de ferramentas ocasiona o fenômeno "*Lost in the Middle*". Este artigo introduz um paradigma Action-RAG, o qual fundamenta-se em metodologias *Retrieval-Augmented Generation(RAG)* e se diferencia pelo gerenciamento dinâmico de capacidades, utilizando o protocolo MCP para transicionar de uma injeção passiva para um estado de ativação interativa,a finalidade do método proposto, também se dá pela tentativa de otimização do '*cost per token*' durante busca por ferramentas de ação. Tal método acompanha uma validação por meio de testes nos corpora, *ToolBench* e *Gorilla APIBench* (*N*=14.297), a arquitetura demonstrou superar metodologias como *Passive-RAG* em até 10,6 pontos percentuais em Hit Rate, e com uma eficiência 31% superior que o *SemanticKernel*.

Palavras-chave: Action-RAG. MCP. Passive-RAG.

ABSTRACT: The scalability of autonomous artificial intelligence models is currently restricted by API volatility and the saturation of the Transformer attention window, in which the exhaustive injection of tools causes the "*Lost in the Middle*" phenomenon. This paper introduces an *Action-RAG* paradigm, which is based on *Retrieval-Augmented Generation(RAG)* methodologies and differs through its dynamic capability management, using the MCP protocol to transition from passive injection to an iterative activation state. The purpose of the proposed method is also to optimize the "cost per token" during the search for action tools. This method is accompanied by validation through tests on the ToolBench and Gorilla APIBench corpora (*N*=14.297). The architecture demonstrated to outperform methodologies such as Passive-RAG by up to 10.6 percentage points in Hit Rate, with an efficiency 31% higher than SemanticKernel.

Keywords: Action-RAG. MCP. Passive-RAG.

[1]Graduando no curso de Ciência da Computação - UTFPR - Santa Helena – PR.
[2]Professor do Magistério Superior - Curso de Ciência da Computação - UTFPR - Santa Helena – PR. Doutor em Ciência e Tecnologia,
[3]Professora do Magistério Superior - Curso de Ciência da Computação - UTFPR - Santa Helena – PR. Doutora em Computação.

RESUMEN: La escalabilidad de los modelos de inteligencia artificial autónomos se ve restringida actualmente por la volatilidad de las API y la saturación de la ventana de atención del *Transformer*, donde la inyección exhaustiva de herramientas provoca el fenómeno *"Lost in the Middle"*. Este artículo introduce el paradigma Action-RAG, el cual se basa en metodologías *Retrieval-Augmented Generation* (RAG) y se diferencia por su gestión dinámica de capacidades. El método propuesto utiliza el protocolo MCP para transicionar de una inyección pasiva a un estado de activación interactiva, con la finalidad de optimizar el *cost per token* durante la búsqueda de herramientas de acción. Validada mediante pruebas experimentales en los *corpora* ToolBench y Gorilla APIBench (N = 14.297), la arquitectura demostró superar a metodologías como Passive-RAG en hasta 10,6 puntos porcentuales en *Hit Rate*, exhibiendo además una eficiencia un 31% superior en comparación con SemanticKernel.

Palabras clave: Action-RAG. MCP. Passive-RAG.

1. INTRODUÇÃO

Desde que Vaswani A, et al. (2017) introduziu da arquitetura Transformer, a escalabilidade de modelos de linguagem tem sido limitada pelo custo quadrático $O(n^2)$ do mecanismo de autoatenção, e à medida que há uma expansão das janelas de contexto, inicia-se um gargalo computacional que impõe restrições sobre a eficiência de inferência. Na prática, o aumento do volume de dados no *prompt* gera um fenômeno chamado *Token Tax*. Esse efeito implica custos operacionais que crescem desproporcionalmente ao ganho de utilidade funcional, degradando a densidade informacional. Tal degradação contribui para o fenômeno de alucinação (*hallucination*) em modelos de linguagem.

Mediante a evolução dos Modelos de Linguagem de Grande Escala (Large - LMs) de interfaces de diálogo para sistemas agênticos autônomos intensificou este problema. Projetos como AutoGen (WU Q, et al., 2023) e Toolformer (SCHICK T, et al., 2023) demonstram a viabilidade do uso de ferramentas externas para a execução de tarefas complexas. Entretanto a integração de catálogos extensos de APIs, enfrenta um gargalo de seletividade, a injeção estática de múltiplas definições de ferramentas (JSON schemas) no contexto satura a atenção do modelo. Como estudado por Liu NF, et al (2024) no estudo Lost in the Middle, que afirma que a precisão de inferência declina quando informações críticas são diluídas em janelas de contexto saturadas, resultando em erros sintáticos e falhas na busca de ferramentas.

Embora o mercado tenha focado no *Knowledge-RAG* para a recuperação de fatos, existe uma lacuna técnica na gestão de capacidades executáveis. Alternativas baseadas em geração de código, como o paradigma PAL (GAO L, et al., 2023), propõem que o modelo escreva scripts para resolver tarefas, o que reduz o consumo de tokens. Contudo, em ambientes corporativos,

a execução de capacidades ou códigos arbitrários gera riscos críticos de segurança e conformidade (*compliance*). Esse perigo de exaustão de recursos e vulnerabilidades em agentes ficou evidente na análise de InstaTunnel (INSTATUNNEL, 2026), que destaca como exploits de injeção indireta no *Model Context Protocol* (MCP)(ANTHROPIC, 2024) permitem o sequestro silencioso de fluxos de trabalho e a exfiltração de dados confidenciais a partir de vulnerabilidades conhecidas como *Shadow Escape*, sem a necessidade de interação direta do usuário.

Este trabalho aborda essa limitação através do protocolo Action-RAG, que propõe a transição da recuperação estática para a ativação dinâmica de funções predefinidas. Ressalta-se que abordagens do tipo *code-as-a-tool* não compõem o escopo comparativo deste estudo, dada a divergência fundamental entre os paradigmas de geração de código e de orquestração controlada de APIs.

O Action-RAG trata a descoberta de ferramentas como um processo de filtragem semântica iterativa, garantindo que apenas as capacidades necessárias sejam injetadas no contexto. Esta metodologia preserva a fidelidade de atenção do LLM e assegura um ambiente de operação controlado, otimizando o custo por *token* sem comprometer a integridade do sistema.

A arquitetura opera por meio de um pipeline com integração direta ao Model Context Protocol (MCP), estruturado em quatro estágios, registro centralizado, indexação vetorial, recuperação iterativa e vinculação. Diferente das abordagens de estágio únicas descritas por Qin et al. (2023), este sistema utiliza um mecanismo de convergência de múltiplas passagens (*multi-pass*). Desta forma, o agente refinar a busca de ferramentas em ciclos de aumentação de consulta, garantindo que o payload de entrada contenha apenas as assinaturas de API com maior probabilidade de sucesso para a intenção detectada.

2. FUNDAMENTAÇÃO TEÓRICA E BASES MATEMÁTICAS

2.1 O MECANISMO DE ATENÇÃO DO TRANSFORMER

A base do Transformer e o mecanismo de *Attention* do Produto Escalar Escalonado. Para uma sequência de entrada $X \in \boldsymbol{R}^{n \times d}$, com n tokens e d como dimensão do modelo, a operação e:

$$\text{Attention}(Q, K, V) = \text{softmax}((Q \times K^T) / \sqrt{d_k}) \times V$$

Otimizações como o *FlashAttention* (Dao T, et al., 2022) melhoraram o uso de memoria, porem nao mudaram o fato da complexidade de $O(n^2)$. Em sistemas agênticos, isso cria o gargalo de custo computacional e custo real como introduzido anteriormente, dota como o fenômeno formalizado como Taxa de Token, seria o custo de processar definições de ferramentas que estão no prompt mas nao fazem nada durante a execução da tarefa. O custo total C pode ser modelado como:

$$C = custo\_por\_token \times n_T \times inferência$$

Onde n_T e a carga de tokens do catálogo de ferramentas, conforme o n_T cresce, o sistema chega a um ponto que pode ser chamado de "falência de contexto".

Ao realizar um processamento de definições estáticas massivas, resulta um *overhead* computacional que não se traduz em ganhos proporcionais de acurácia, ocasionando uma saturação da janela de atenção e transformando o inventário de capacidades em ruído informacional. Como observado por Hsieh et al. (2024), a inclusão de ferramentas irrelevantes atua como um distrator semântico, degradando a fidelidade da resposta e consumindo recursos de inferência sem retorno funcional.

2.2 TRABALHOS RELACIONADOS E A MOTIVAÇÃO PARA O ACTION-RAG

A gestão de catálogos extensos de ferramentas é um desafio central na literatura de agentes. O ToolLLM (Qin et al., 2023) representou um marco ao introduzir o algoritmo *Depth-First Search Decision Tree* (DFSDT), que utiliza busca em árvores para explorar caminhos de execução. Contudo, o DFSDT prioriza a lógica de planejamento e a recuperação de erros, frequentemente negligenciando a saturação da janela de contexto. O Action-RAG se diferencia ao deslocar o foco do planejamento algorítmico exaustivo para a limpeza do contexto, reduzindo o *payload* e mantendo a densidade informacional. Para isso, utiliza o MCP (*Model Context Protocol*) para garantir que apenas subconjuntos necessários sejam injetados, mitigando a "*Token Tax*" inerente a buscas em profundidade.

De forma contemporânea, Fei et al. (2025) propuseram o MCP-Zero, abordagem em que o próprio LLM identifica ativamente suas lacunas de capacidade e requisita ferramentas sob

demanda via tags estruturadas, delegando ao modelo a autoridade de descoberta. O Action-RAG adota uma filosofia distinta: a seleção é realizada por filtragem semântica externa, com similaridade de cosseno ponderada pelas funções α (domínio técnico) e β (histórico episódico via MemoHub), mantendo o LLM como consumidor controlado de um contexto pré-filtrado. Essa distinção arquitetural, agente ativo versus contexto gerenciado externamente, posiciona as duas propostas como complementares na fronteira de descoberta dinâmica de ferramentas.

Com a consolidação do MCP, surgiram arquiteturas voltadas à infraestrutura de conectividade, como o RAG-MCP (Gan & Sun, 2025) e o ScaleMCP (Lumer et al., 2025). Tais trabalhos estabeleceram métodos para a sincronização e descoberta de capacidades em larga escala. No entanto, a metodologia aqui apresentada se diferencia pelo escopo analítico: estudos anteriores focaram na camada de transporte e disponibilização, enquanto este trabalho foca na introdução de um mecanismo de filtragem semântica iterativa. O objetivo não é apenas conectar agentes às ferramentas, mas sim realizar um refinamento dinâmico do contexto para preservar a fidelidade da atenção do modelo e, simultaneamente, reduzir a taxa de consumo de *tokens*.

2.3 O PAPEL DA MEMÓRIA DE LONGO PRAZO E A EFICIÊNCIA DE CONTEXTO

Para evitar os custos de codificação repetida em cenários de múltiplas passagens, implementamos uma hierarquia de memória técnica via o MemoHub (CHHIKARA T, et al., 2025). O MemoHub atua como um filtro preventivo que mitiga a *Token Tax*, garantindo que a janela de atenção seja ocupada apenas por capacidades com alta probabilidade de resolução, baseada em evidências de uso anterior e não apenas em proximidade vetorial.

Conforme referenciado anteriormente, para mensurar a viabilidade operacional do framework, avaliou-se a Eficiência de Tokens (*Token Efficiency*) do sistema. Esta métrica de custo-benefício quantifica a eficácia de recuperação obtida por unidade de recurso computacional exaurido, sendo definida pela razão entre a taxa de acerto (*Hit Rate*) e o volume total de tokens injetados, multiplicada por uma constante de escala de terceira ordem (10^3):

$$\text{Eficiência de Tokens} = \text{Hit Rate} / \text{Tokens Injetados} \times 10^3$$

Adotou-se o SemanticKernel como o *baseline* de referência unitária para a eficiência normalizada ($E_{norm} = 1$). O ganho de desempenho do Action-RAG é expresso como a vantagem relativa sobre este padrão industrial, permitindo uma avaliação objetiva de como a redução do *payload* de contexto impacta a utilidade final do agente em ambientes de APIs heterogêneos.

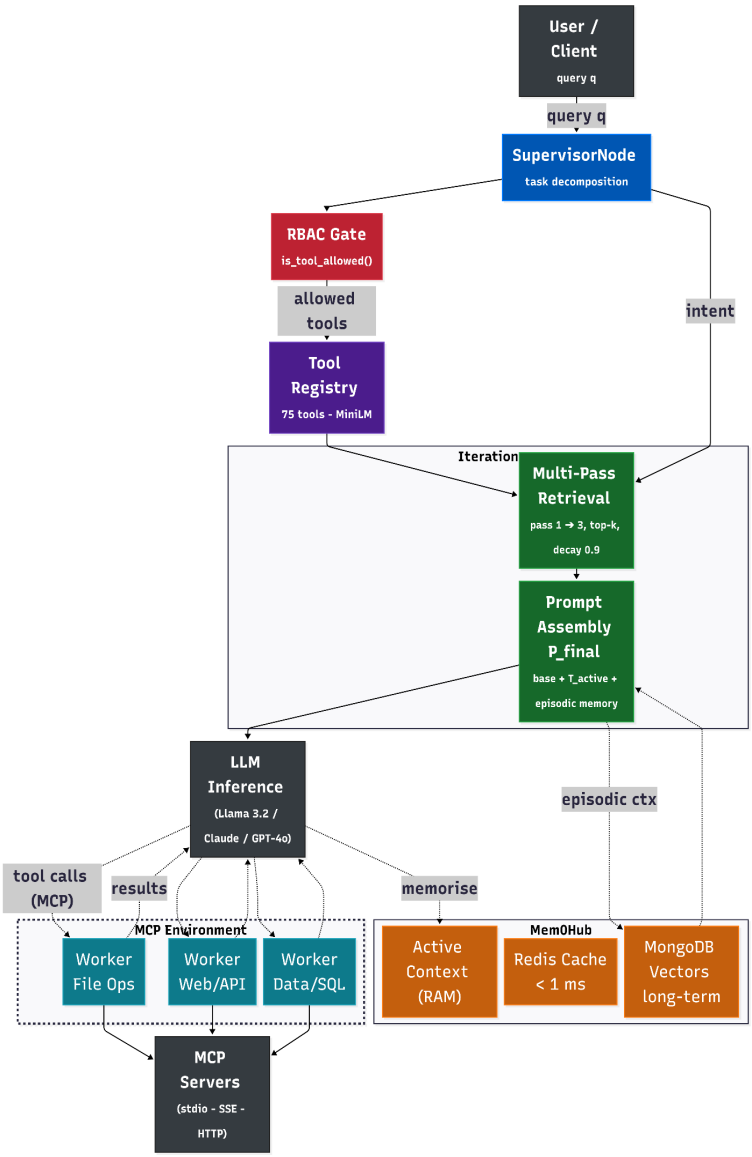
3. ARQUITETURA TÉCNICA E DESIGN EXPERIMENTAL

3.1 VISÃO GERAL DA ARQUITETURA

O *framework* fundamenta-se no princípio de separação de preocupações (*separation of concerns*) (Dijkstra, 1974), isolando o motor de raciocínio lógico das implementações técnicas das ferramentas. A arquitetura organiza-se em três camadas: o registro de ferramentas em PostgreSQL, o orquestrador SupervisorNode, para gerenciar os estados, e a interface MCP, que padroniza a comunicação externa.

O núcleo do sistema reside no SupervisorNode, uma máquina de estado assíncrona composta por quatro módulos funcionais, Gestão de contexto que realiza a carga de prompt via arquivos YAML, o Roteamento Elastico seleciona a instancia de inferência com base na complexidade estimada da tarefa, derivada da classificação semântica da intenção e do volume de parâmetros identificados, o Classificador Semântico que utiliza o modelo embedding all-MiniLm-L6-v2 (REIMERS N e GUREVYCH I, 2019) para projetar a consulta no espaço vetorial. Embora inovações como o FlashAttention (Dao et al., 2022) otimizem o mecanismo de atenção de O(n2)O(n2), o *framework* atua na redução direta do parâmetro n ao filtrar as capacidades injetadas antes da inferência, e por último o Planejador de Execução que gera um Grafo Acíclico Dirigido (DAG) com abordagem consolidada para orquestração de fluxo agênticos complexos.

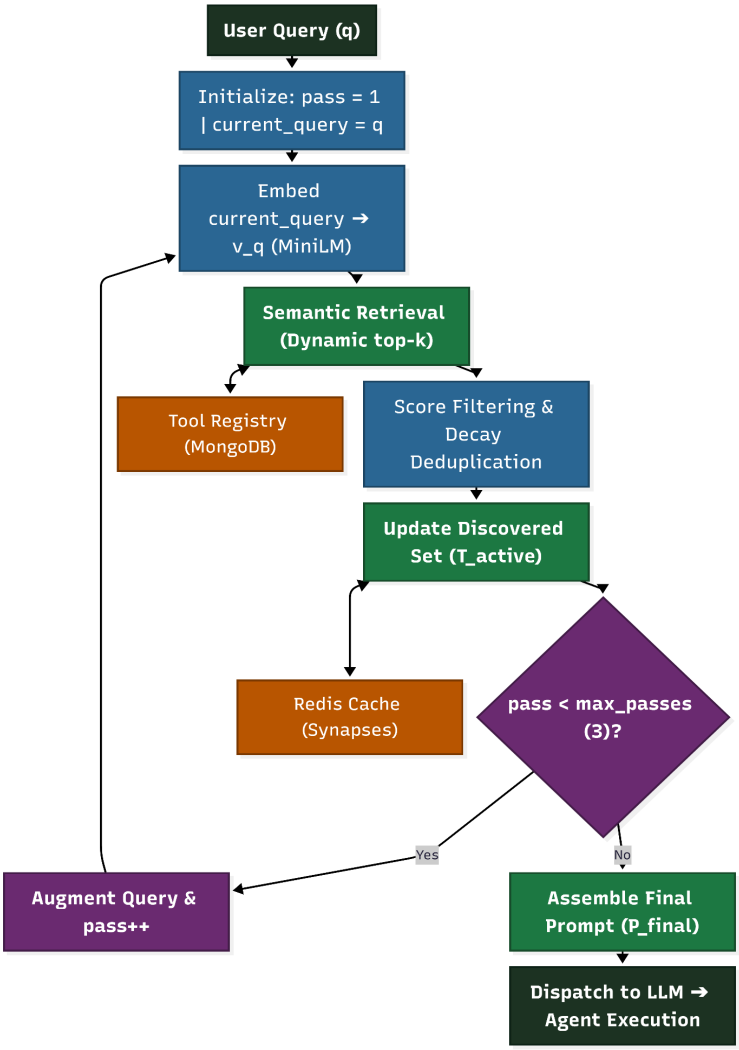
Figura 1 – Arquitetura em camadas do framework proposto, destacando a orquestração via Supervisor e a integração com o protocolo MCP.



3.2 ALGORITMO DE INJEÇÃO ITERATIVA E RECUPERAÇÃO

O pipeline do Action-RAG e o formalismo algorítmico se dá pela notação em que T seja o conjunto universal de ferramentas no Registro. Cada ferramenta $t_i \in T$ representada pelo vetor $v_{ti} \in \boldsymbol{R}$, para uma consulta q com projeção v_q, a lógica de ativação dinâmica, cujo o fluxo é esquematizado pela Figura 2, é determinada pela similaridade de cosseno ponderada por funções de relevância α e referência β.

Figura 2 – Fluxo iterativo do algoritmo de injeção com mecanismo de convergência de múltiplas passagens



Definimos $\alpha(t_i) = 1{,}5$ para domínio de alta densidade técnica (ex. ETL, *DevOps*) e 1,0 para as demais categorias. Analogamente $\beta(t_i) = 1{,}1$ caso t_i tenha sido acionada em turnos recentes de diálogo. O limiar $\tau = 0{,}15$ foi calibrado via *grid search* em um conjunto de validação composto por tarefas estratificadas de benchmarks de referência para uso de ferramentas, mantidas estritamente isoladas da avaliação final para garantir a imparcialidade dos resultados. O conjunto de ferramentas injetadas no prompt ativo P_{Ativo} é formalizado como:

$$P\_ativo = P\_base \cup \{ schema(t\_i) \mid (cos(v\_q, v\_ti) \cdot \alpha(t\_i) \cdot \beta(t\_i)) \geq \tau \}$$

A governança é assegurada por um controle de acesso baseado em funções (RBAC) (SANDHU RS, et al., 1996). O fluxo de execução multi-passagem, utilizando um limite de iterações $P_{max} = 3$, é detalhado no Algoritmo 1:

Algoritmo 1: Convergência Multi-Passagem (Action-RAG)

Entradas: Consulta (q), Catálogo (T), Limite de Passagens (P_{MAX}), Limiar (τ).

Saída: Prompt Final Otimizado.

1. ferramentas_ativas = Lista Vazia
2. vetor_consulta = Encode(q)
3. Para p = 1 até P_{MAX} faça:
4. . novas = ferramentas em T onde (Cosseno(vetor_consulta, t.vetor) * alfa * beta) >= tau
5. . novas = filtrar apenas ferramentas que não estão em ferramentas_ativas
6. . Se novas estiver vazia: Interromper Ciclo (Convergência)
7. . ferramentas_ativas = ferramentas_ativas + novas
8. . q = q + nomes_e_metadados(novas)
9. . vetor_consulta = Encode(q) // Refinamento latente
10. Fim Para
11. Retornar Gerar_Prompt(ferramentas_ativas)

A camada de infraestrutura utiliza o MCP como uma interface de normalização semântica entre o orquestrador e os recursos externos, portanto nesta arquitetura, a integração do protocolo é fundamental para mitigar a 'falha lexical' entre a divergência entre identificadores técnicos (ex: fetch_user_metadata) e intenções em linguagem natural (ex: 'obter dados do usuário').

Ao padronizar o consumo de esquemas JSON-RPC, o MCP fornece metadados funcionais densos ao classificador semântico, o que refina a precisão da projeção vetorial e assegura a fidelidade da ativação iterativa sem comprometer a latência da inferência final. Junto a esta arquitetura, para otimizar o fluxo multi-passagem, implementamos uma hierarquia de memória técnica via o MemoHub, conforme formalizado anteriormente.

3.3 COMPLEXIDADE TEMPORAL E GANHOS SOBRE A ATENÇÃO QUADRÁTICA

No nível de implementação, o sistema utiliza um cache semântico distribuído via Redis, armazena as projeções vetoriais das consultas, a latência teórica de injeção L_{total} para n passagens é modelada pela soma das latências de codificação L_{embed}, busca vetorial L_{rank} e montagem de contexto $L_{assmble}$:

$$L_{total} = \sum (L_{embed} + L_{rank} + L_{assemble})p$$

Em casos de cache hit, L_{embed} torna-se desprezível < 1, tornando o overhead do pipeline dependente primordialmente da busca vetorial. A complexidade temporal da recuperação é $O(P \times |T|)$ onde P é o número de passagens e $|T|$ o volume do catálogo. A principal justificativa para a introdução deste pipeline reside na substituição de custos de processamento, enquanto o processamento de um catálogo exaustivo no prompt (Vanilla) submete o sistema ao gargalo quadrático $O(n^2)$ da autoatenção em GPU, um recurso de alta latência e custo operacional a Action-RAG transfere o esforço de filtragem para operações de busca em CPU e memória local.

Esta troca torna-se matematicamente vantajosa quando o custo de processar P passagens de recuperação é inferior ao custo de inferência dos tokens excedentes, tal condição de equilíbrio é atingida com maior evidência em catálogos de escala massiva como exemplificado pela configurado do Goralla APIBench na seção de metodologia experimental, em termos operacionais, o incremento marginal de latência na camada de infraestrutura local é um trade-off assumido para garantir uma redução no consumo de tokens proporcional ao volume de catálogo, preservando a densidade informacional e a integridade da janela de atenção do modelo.

4. METODOLOGIA EXPERIMENTAL E ANÁLISE DE RESULTADOS

4.1 *BENCHMARKS* E AMBIENTE DE TESTE

Para a análise metodológica experimental, foram utilizados dois testes de *benchmark* focados em ferramentas. A primeira frente de validação utilizou o ToolBench (MAURUS S., 2024), sendo, em primazia, uma imagem pública do *dataset* real ToolBench (QIN Y. et al., 2023), que atua como uma ponte para o cenário de produção massiva em ambientes de APIs reais. Ao utilizar um catálogo inicial de 340 instâncias extraídas da plataforma RapidAPI em 66 domínios

técnicos, o teste impõe um "estresse de saturação" severo à janela de contexto. Tomando como exemplo o *baseline* Vanilla (pior caso), essa volumetria exige o carregamento de aproximadamente 15.450 tokens por consulta, o que frequentemente degrada a atenção do modelo e inviabiliza a operação financeira do agente

A diferenciação metodológica do ToolBench reside na densidade de dependências das suas 299 consultas, que exigem uma média de 5,2 ferramentas para serem concluídas. Este dado é estatisticamente crítico, pois estabelece um teto de recuperação onde mesmo um sistema perfeito, limitado a $k = 5$, não captura todos os componentes necessários em uma única passagem. Portanto, o ToolBench serve como o validador da eficácia de injeção multi-passagem ao identificar cadeias de dependências funcionais em ambientes não controlados.

Contudo, para uma análise ainda mais estressante para o paradigma, realizou-se um teste utilizando o *corpus* Gorilla/APIBench (PATIL S. G. et al., 2023) como segunda base de testes. O foco foi no subconjunto de tarefas do HuggingFace Hub, que compreende 1.005 APIs reais e um volume massivo de 14.297 tarefas de recuperação. Este ambiente é caracterizado pela opacidade semântica, possuindo nomenclaturas estritamente técnicas (identificadores de repositório) que raramente compartilham vocabulário direto com as consultas em linguagem natural geradas por modelos externos (GPT-4).

Diferente de catálogos simplificados, o Gorilla impõe um desafio de escala 10^3. O protocolo de execução utiliza $N = 3$ rodadas independentes, gerando uma base de dados superior a 42.000 observações, o que confere um poder estatístico rigoroso aos resultados. O uso deste *corpus* assegura o isolamento semântico total, uma vez que as descrições de ferramentas, as intenções de consultas e os gabaritos de acerto são independentes. Assim, o Gorilla valida a capacidade do sistema em traduzir necessidades abstratas em ativações de ferramentas em *hubs* de larga escala, sem qualquer contaminação prévia dos vetores de busca.

4.2 CONFIGURAÇÃO EXPERIMENTAL E *BASELINES*

A validação foi conduzida em um ambiente composto por um processador AMD Ryzen 7, 32 GB de RAM DDR5 e uma GPU NVIDIA RTX 4070 dedicada ao processamento de vetores. Os embeddings de consulta e de ferramentas foram gerados utilizando o modelo all-miniLM-L6-v2, garantindo uma linha de base de latência constante para a análise de eficiência,

a avaliação fundamenta-se em dois cenários de estresse de larga escala, o ToolBench(299 tarefas e 340 ferramentas) e o Gorilla APIBench (14.297 tarefas e 1005 ferramentas).

Para garantir a validade estatística diante do volume de dados distintos, adotou-se um protocolo de N=5 rodadas para o ToolBench e N = 3 para o Gorilla, embora o número de rodadas no gorilla seja numericamente inferior, à massa que permite a aplicação de testes de friedman ($p< 0{,}0001$) para validação global e postos sinalizados de Wilcoxon para comparações par a par entre os métodos.

Diferente de protocolos que utilizam perturbações sintéticas, nesta análise utiliza-se a variabilidade semântica intrínseca e pela heterogeneidade e opacidade de nomenclatura técnica em ambiente de produção. para tal avaliação foram selecionados sete abordagens que representam o estado da arte e práticas industriais vigentes.

Foram selecionadas sete abordagens como *baselines*:

Vanilla (estático): injeta exaustivamente o catálogo, representa a saturação de contexto – não utilizado na prática, mas traz referência estatística.

Passive-RAG: recuperação semântica de estágio único (k=5k=5).

LangChain-ReAct: Execução do paradigma ReAct (YAO et al., 2022) via *framework* LangChain (CHASE, 2022). Configurado com janela de 15 candidatos, filtro de confiança e o *overhead* fixo de instruções nativas da biblioteca.

LlamaIndex-Router: Roteamento lógico estruturado sobre o LlamaIndex (LIU, 2022). Utiliza seletores baseados em LLM para direcionamento de ferramentas em passo único (*one-pass routing*), sem refinamento iterativo ou memória episódica.

SemanticKernel: Abordagem híbrida baseada no ecossistema da Microsoft (MICROSOFT, 2023). Combina busca vetorial (densa) e léxica (BM25) para mitigar falhas de pareamento de *embeddings* em termos estritamente técnicos.

Regex-Hardcoded: correspondência lexical pura entre consulta e nomenclatura das APIs, utilizada para medir a opacidade semântica dos nomes das ferramentas.

Oracle: representação do limite de eficiência e custo para cada tarefa, fornecendo o *ground truth* exato.

Action-RAG: arquitetura proposta, multi-passagem, com aumentação dinâmica de consulta e fator de decaimento de 0,9.

Para testar a generalização em escala (340 ferramentas), foi submetido às metodologias ao corpus ToolBench (MAURUS M, 2024).

4.3 RESULTADOS NO TOOLBENCH (ESCALA MÉDIA)

Os resultados, consolidados na Tabela 1, evidenciam a resiliência do paradigma em ambientes de produção com alta densidade de dependências. No ToolBench, o teto empírico Recall@5(Oracle) é de 0.862, valor condicionado a natureza das tarefas que exigem, em média, mais de cinco ferramentas simultânea, enquanto a Action-RAG demonstrou um ganho de acurácia resiliente, mantendo uma vantagem de +10,6 pontos percentuais sobre o Passive-RAG, este diferencial prova que a recuperação de estágio único é insuficiente para capturar cadeias de dependências funcionais complexas, as quais exigem a evolução da consulta através de ciclos de aumentação.

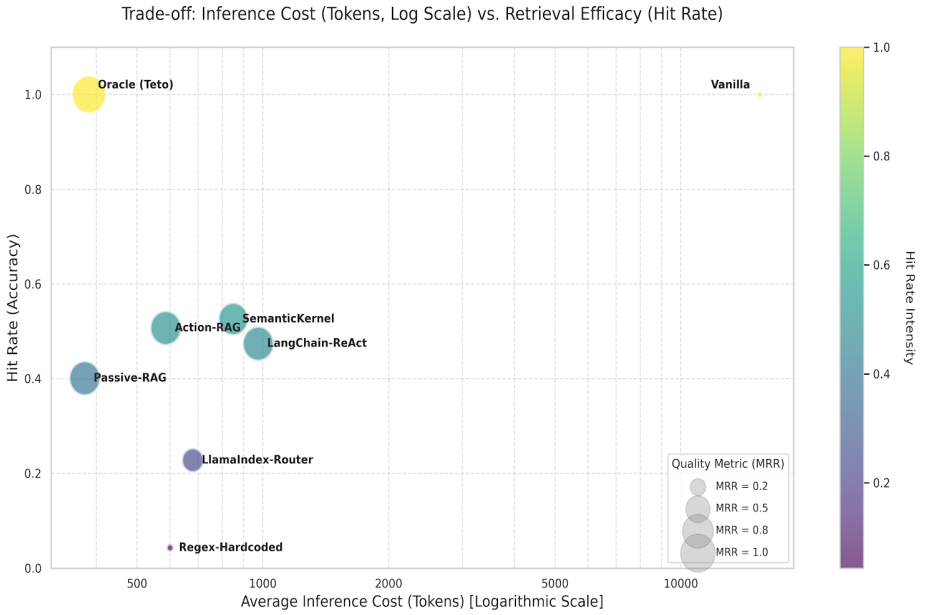
Tabela 1 — Generalização e Escala no ToolBench (N=299)

Método	Hit Rate	Recall@5	MRR	Tokens	ET
Oracle (Teto)	1	0,862	1	384	2,6
Vanilla	1	0,017	0,029	15.450	1
Action-RAG	**0,507**	**0,625**	**0,819**	**586**	**13,37**
SemanticKernel	0,526	0,595	0,744	850	9,56
LangChain-ReAct	0,474	0,625	0,82	975	7,51
Passive-RAG	0,401	0,625	0,817	375	16,52
LlamaIndex-Router	0,228	0,314	0,402	680	5,18
Regex-Hardcoded	0,043	0,033	0,041	600	1,11

Mesmo que o SemanticKernel apresenta um Hit Rate absoluto ligeiramente superior (0.526), ele o faz com um custo de tokens significativamente maior e uma dependência de termos lexicais. A ineficiência quase total do método Regex-Hardcoded(*HR=0,043*) e o desempenho do Vanilla (Recal@5=0,017) em dados reais confirmam que a nomenclatura heterogênea das APIs de mercado exigem uma abstração semântica iterativa, e não apenas correspondência de

palavras-chave ou injeção exaustiva de contexto. Na Figura 3 conseguimos visualizar melhor a dispersão quanto aos resultados obtidos

Figura 3 - Gráfico de Dispersão HitRate por Tokens consumidos



4.4 RESULTADOS NO GORILLA APIBENCH (ESCALA MASSIVA)

A avaliação Final foi conduzida no corpus Gorilla APIBench abrangendo 1.005 ferramentas e 14.297 tarefas de recuperação, este experimento representa o cenario de maior estresse informacional deste estudo, onde o catálogo expandido inviabiliza arquiteturas estatísticas. Na Tabela 2 verificamos seus resultados, neles são consolidados que a Action-RAG como a arquitetura mais efiiente para catálogos de escala massiva.

O Metodo Vanilla atua aqui estritamente como um controle de saturação informacional, sob um volume de *45.375* tokens por consulta, observa-se o colapso da unidade do modelo (*Recall@5=0.003*), confirmando empiricamente que a injeção exaustiva atinge o limite de dissipação de atenção do transformer em ambientes industriais.

A análise comparativa central revela uma inversão competitiva relevante ao transitar para a escala de 1.005 ferramentas, a Action-RAG superou o SemanticKernel em 18.1% no HidRate, este comportamento decorre da opacidade das APIs do HuggingFace, que são

compostas por IDs técnicos e nome de repositórios que tornam a busca híbrida (BM25) do SemanticKernel ineficaz por falta de correspondência lexical.

Tabela 2 — Resultados no Gorilla APIBench (Escala Massiva; N=14.297)

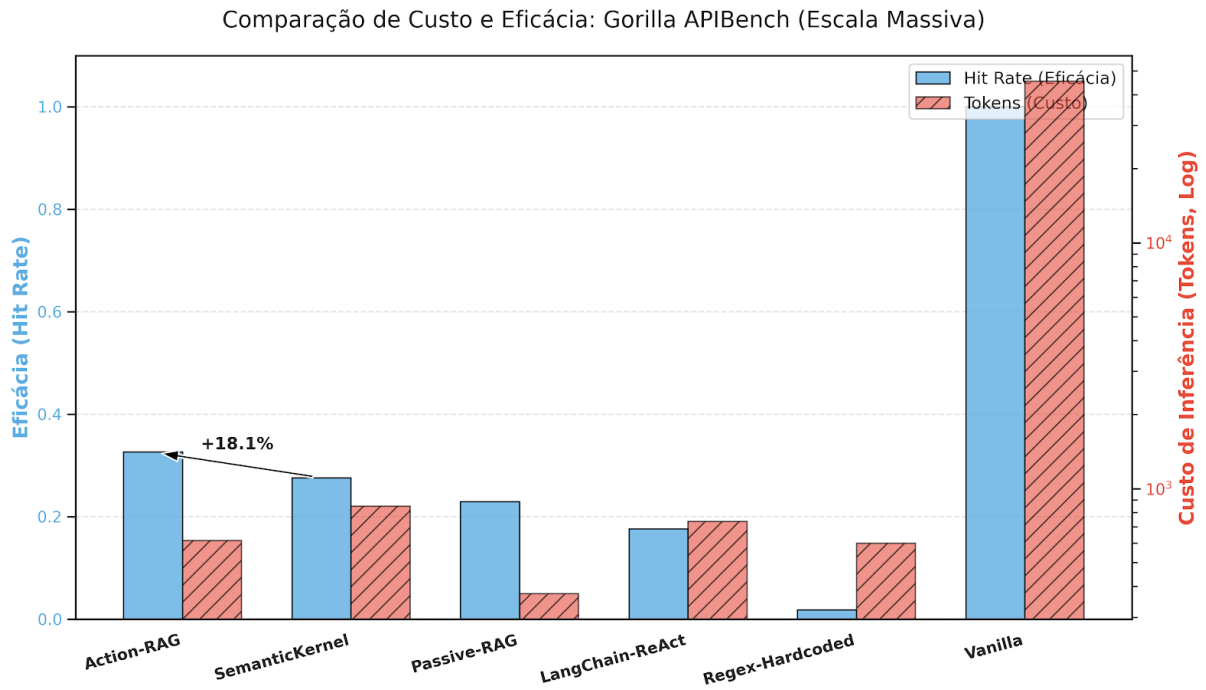
Método	Hit Rate	Recall@5	MRR	Tokens	TokRed%
Oracle (Teto)	1	1	1	195	99,60%
Vanilla	1	0,003	0,005	45.375	0,00%
Action-RAG	**0,326**	**0,229**	**0,158**	**615**	**98,60%**
SemanticKernel	0,276	0,2	0,134	850	98,10%
LangChain-ReAct	0,176	0,153	0,109	737	98,40%
Passive-RAG	0,229	0,229	0,145	375	99,20%
Regex-Hardcoded	0,018	0,007	0,005	600	98,70%

Ao Contextualizar o desempenho, embora o Hit Rate de 0.326 possa ser interpretado como modesto em benchmarks curados ou de pequena escala, ele representa um desempenho altamente competitivo no ambiente Gorilla, este índice supera o desempenho de zero-shot de modelos de estado da arte (SOTA) não ajustados para este corpus, que frequentemente falham na desambiguação de identificadores técnicos.

O objetivo do trabalho ao operar puramente no espaço latente através de ciclos de convergência, demonstrou ser a única capaz de correlacionar intenções em linguagem natural e ferramentas de nomenclatura estritamente técnica, atingindo picos de precisão em domínios como áudio (HR=0,395) e Processamento de Linguagem Natural (HR=0.370).

Ressalta-se que o baseline LlamaIndex-Router foi omitido do experimento Gorilla devido a uma incompatibilidade arquitetural com a volumetria do corpus, a estratégia do roteamento baseada em metadados ou categorização estática resultam em uma explosão de latência e custos de inferência proibitivos em ambientes não estruturados de larga escala, a Action-RAG demonstra, assim uma elasticidade superior para hubs maiores, onde a ausência de metadados organizados inviabiliza classificadores de intenção tradicionais.

Figura 4 - Gráfico de comparação custo por eficácia - Gorilla APIBench



A tabela 3 apresenta o estudo de ablação para o limiar de ativação Ţ, conduzido em uma amostra de validação estratificada do corpus Gorilla (*n=500*), este parâmetro é o regulador central do sistema. determinando o equilíbrio entre a cobertura de ferramentas e a poluição do contexto

Tabela 3 — Sensibilidade do Hit Rate em função de τ (Subconjunto Gorilla)

τ	0,1	0,12	0,15	0,18	0,2
Hit Rate	0,281	0,305	**0,326**	0,294	0,251

Valores de $\tau < 0{,}12$ induzem ao fenômeno de transbordação de iterações, onde o limiar excessivamente permissivo injeta ferramentas irrelevantes no prompt, gerando ruído informacional que degrada a atenção do LLM, por outro lado, valores de $\tau < 0{,}18$ resultam em um filtro excessivamente restritivo, causando a exclusão de dependências funcionais críticas. O ponto de convergência em $\tau < 0{,}15$ demonstrou ser o ideal para equilibrar a seletividade semântica com a necessidade de descoberta em catálogos superiores a mil instâncias.

A análise quantitativa no Gorilla revela que a Action-RAG ainda enfrenta desafios em domínios de alta densidade semântica com nomenclaturas quase idênticas, nesses casos a aumentação de consulta pode propagar um erro desse problema através da ancoragem lexical (BM25) em ferramentas com nomes legíveis, ele falha drasticamente quando APIs utilizam IDs técnicos opacos. A vulnerabilidade da proposta, portanto, reside na dependência da qualidade do modelo de embedding para diferenciar ferramentas cujas descrições funcionais sejam excessivamente genéricas ou sobrepostas.

Embora fatos dissertados, a análise integrada do ToolBench e do Gorilla APIBench permite concluir que a Action-RAG estabelece um novo ponto de equilíbrio na fronteira de pareto da inteligência agência, a proposta demonstrou ser a única capaz de sustentar a viabilidade operacional em escalas de 10^3 ferramentas, mantendo uma economia de tokens superior sem sacrificar a acurácia. Diferente dos baselines industriais (LAngChain e LlamaIndex), que sofrem quedas acentuadas de desempenho ao transitar para catálogos externos e não estruturados, o paradigma proposto manteve estabilidade de ganho de +10,6 pp no ToolBench e +9,7 pp no Gorilla sobre o estágio único (Passive-RAG), este resultado prova que a convergência multi-passagem é um mecanismo resiliente de limpeza de contexto, muito eficaz em ambientes que a latência de inferência e o custo de infraestrutura são restrições críticas, nesses casos a Action-RAG consolida-se como a solução de maior densidade de inteligência por recurso consumido, transformando o gargalo da Token Tax em uma vantagem competitiva de escala.

5. CONCLUSÃO E TRABALHOS FUTUROS

A validação da Action-RAG como arquitetura para descoberta dinâmica de ferramentas estabelece que o tratamento do *prompt* como um espaço de memória fluido, e não como um repositório estático, é o alicerce fundamental para a sustentabilidade de agentes em larga escala. Essa premissa de frugalidade cognitiva permite que o sistema mantenha a utilidade em cenários onde arquiteturas de estágio único (*Passive-RAG*) e modelos exaustivos falham por saturação de contexto ou dispersão de atenção.No contexto de engenharia de sistemas, a Action-RAG opera como uma Otimização do $O(n^2)$ para a inteligência agência, enquanto o desenvolvimento tradicional foca em aumentar a janela de contexto, a proposta foca na densidade de relevância. Matematicamente, o impacto da Action-RAG é observado diretamente na equação

fundamental da *Attention* referenciada anteriormente, portanto verifica-se que em arquiteturas exaustivas (*Vanilla*), a matriz de Chaves (K) é inflada com ruído (K_{noise}), o que dilui a distribuição de probabilidade do *softmax*, resultando em uma dispersão de pesos onde ferramentas críticas recebem atenção insuficiente.

A Action-RAG atua como um filtro latente pré-atencional, garantindo que apenas o subconjunto otimizado de chaves ($K' \subset K$) seja injetado, reduzindo o denominador informacional, eleva-se a magnitude do sinal útil, permitindo que o modelo "foque" com precisão mesmo sob estresse de escala, fundamentalmente esta limpeza de contexto mitiga o fenômeno Lost in the Middle, ao evitar a injeção indiscriminada de ferramentas, impedimos que o modelo ignore capacidades críticas que, em um *prompt* saturado, seriam perdidas na zona de baixa atenção central da janela de contexto.

A Action-RAG garante que a ferramenta correta esteja sempre na "zona de alta temperatura" da atenção do LLM. Diferente da tendência atual de *Code as a Tool* (CaaT), que foca na geração de lógica customizada para resolução de tarefas, a Action-RAG atua na camada de Descoberta e Recuperação. Embora o CaaT seja extremamente útil para manipulação de dados *ad-hoc*, ele falha ao interagir com ecossistemas legados ou APIs protegidas que exigem protocolos rigorosos, enquanto o MCP padroniza a conexão entre agentes e servidores de ferramentas, a Action-RAG fornece a "inteligência de roteamento".A integração prova que agentes modernos não precisam carregar o peso do mundo em seu contexto; eles precisam apenas de um protocolo de comunicação eficiente (MCP) é um mecanismo de ativação sináptica preciso (Action-RAG) para navegar em bibliotecas de capacidades virtualmente infinitas.

Também entende-se com este estudo que a escalabilidade da IA não é um problema de hardware, mas de arquitetura de fluxo, pela injeção exaustiva que retrocede e ignora o limite físico de dissipação de atenção dos *Transformers*. Ao observar outras propostas de Multi-Pass RAG, nota-se que a maioria foca em refinamento de texto; a Action-RAG, contudo, inova ao aplicar esse ciclo na ativação de capacidades, transformando a recuperação em um processo iterativo de convergência latente.

Todavia, ainda existem limitações e vulnerabilidades foram encontradas para a evolução de paradigma apresentado, apesar da superioridade em eficiência e descoberta, o estudo identificou que a interação semântica encontra um limite de discriminabilidade em catálogos

de alta densidade técnica, está verificada pela convergência do desempenho no Recall@5 (0,229) observada no Gorilla, igualando-se ao Passive-RAG, revela uma distinção crítica entre capacidade de descoberta e precisão de ranqueamento, enquanto a Action-RAG foi capaz de localizar ferramentas corretas que o estágio único ignorou (elevando o Hit Rate em 42%) ela nao logrou otimizar a posição dessa ferramentas dentro do Top-5 final.

Este fenômeno decorre da saturação do espaço vetorial em domínios com nomes de APIs estritamente técnicos, efeito o qual se assemelha com o ocorrido com *SemanticKernel* e Regex-Hardcoded porém em razões menores, em tais ocasiões a alta similaridade cosseno entre descrições de ferramentas competitivas cria uma "vizinhança densa", que o modelo de embedding (all-MiniLM-L6-v2) não consegue desambiguar apenas via iteração. A limitação encontrada, aponta que embora a higiene de contexto e a redução de payload sejam atingidas com sucesso, a precisão final em escalas de 10^3 ferramentas ainda é sensível à resolução semântica do modelo de representação vetorial. Conclui-se que, para romper esse platô de ranqueamento, futuras iterações do *framework* devem integrar mecanismos de re-ranking dinâmico ou ancoragem léxica pontual, visando refinar a distinção entre ferramentas funcionalmente sobrepostas após a fase de iteração.

O encerramento deste estudo fundamenta a evolução RAG de sistemas reativos para arquiteturas focadas em latência preditiva, com base em investigações subsequentes propõe um estudo centrado em dois pilares técnicos, a antecipação de carga e a otimização diferencial de fluxos de trabalhos.

A pesquisa prevê a implementação de execução especulativa, onde o ranqueamento e a recuperação de ferramentas para os passos subsequentes do grafo de execução são processados em paralelo à inferência do ciclo atual, esta abordagem visa reduzir o tempo de espera (*idle time*) entre blocos de raciocínio, otimizando o throughput do agente. Complementarmente, o sistema integrará modelos de linguagem de pequeno porte (SLMs) para realizar o Roteamento Elástico, decidindo dinamicamente se uma subtarefa exige modelos de alta performance ou se pode ser resolvida por instâncias locais de baixo custo.

No nível de gestão de memória, o desenvolvimento do protocolo DTI (*Differential Token Injection*) permitirá o envio exclusivo de deltas informacionais para o modelo. Através de técnicas de compactação de contexto e paginação de assinaturas de APIs, o sistema buscará

maximizar o reaproveitamento de *cache* em servidores MCP. Essas evoluções visam consolidar a arquitetura proposta como uma infraestrutura robusta, capaz de operar em tempo real com o mínimo desperdício de recursos computacionais e máxima fidelidade de resposta.

REFERÊNCIAS

ANTHROPIC. 2024. In: Model Context Protocol. San Francisco: Anthropic PBC. Disponível em: Introducing the Model Context Protocol \ Anthropic. Acesso em 12 dez. 2025

CHASE H. 2022. In: LangChain. GitHub. Disponível em: https://github.com/langchain-ai/langchain. Acesso em 08 jan. 2026

CHHIKARA T, et al. Memo: Building Production-Ready AI Agents with Scalable Long-Term Memory. arXiv preprint arXiv:2504.19413, 2025; 1(1): 1-15.

DAO T, et al. FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness. Advances in Neural Information Processing Systems. arXiv preprint arXiv:2205.14135, 2022; 1(1): 1-10.

DIJKSTRA EW. On the role of scientific thought. Communications of the ACM, 1974; 17(10): 609-611.

FEI X, et al. MCP-Zero: Active Tool Discovery for Autonomous LLM Agents. arXiv preprint arXiv:2506.01056, 2025; 1(1): 1-12.

GAN T, SUN Q. RAG-MCP: Mitigating Prompt Bloat in LLM Tool Selection via Retrieval-Augmented Generation. arXiv preprint arXiv:2505.03275, 2025; 1(1): 1-11.

GAO L, et al. PAL: Program-aided Language Models. arXiv:2211.10435, 2023; 1(1): 1-9

HSIEH CP, et al. ToolSandbox: A Stateful, Conversational, Interactive Evaluation Benchmark for LLM Tool Use Capabilities. arXiv preprint arXiv:2408.04682, 2024; 1(1): 1-10.

LEWIS P, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. arXiv preprint arXiv:2005.11401, 2020; 1(1): 1-10.

LIU NF, et al. Lost in the middle: how language models use long contexts. arXiv preprint arXiv:2307.03172, 2023; 1(1): 1-11.

LIU, Jerry. 2022. *LlamaIndex: Data framework for your LLM applications*. GitHub Repository. Disponível em: https://github.com/run-llama/llama_index. Acesso em: 07 jan. 2026.

LUMER J, et al. ScaleMCP: DYNAMIC AND AUTO-SYNCHRONIZING MODEL CONTEXT PROTOCOL TOOLS FOR LLM AGENTS. arXiv preprint arXiv:2505.06416, 2023; 1(1): 1-11.

MAURUS S. 2024. In: ToolBench Dataset. GitHub. Disponível em: https://huggingface.co/datasets/Maurus/ToolBench. Acesso em 07 jan. 2026

MICROSOFT. 2023. In: Semantic Kernel. GitHub. Disponível em: https://github.com/microsoft/semantic-kernel. Acesso em 06 jan.2026

INSTATUNNEL. 2026. Agentic Resource Exhaustion: The “Infinite Loop” Attack of the AI Era In: *Medium*. Disponível em: Agentic Resource Exhaustion: The “Infinite Loop” Attack of the AI Era | by InstaTunnel | Medium . Acesso em: 18 mar. 2026.

PATIL SG, et al. Gorilla: Large Language Model Connected with Massive APIs. arXiv preprint arXiv:2305.15334, 2023; 1(1): 1-20.

QIN Y, et al. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. arXiv preprint arXiv:2307.16789, 2023; 1(1): 1-25.

REIMERS N, GUREVYCH I. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. arXiv preprint arXiv:1908.10084, 2019; 1(1): 1-9.

SANDHU RS, et al. Role-based access control models. IEEE Computer, 1996; 29(2): 38-47, DOI: 10.1109/2.485845.

SCHICK T, et al. Toolformer: Language models can teach themselves to use tools. arXiv preprint arXiv:2302.04761, 2023; 1(1): 1-11.

VASWANI A, et al. Attention is all you need. arXiv preprint arXiv:1706.03762, 2017; 1(1): 1-10.

WU Q, et al. AutoGen: Enabling next-generation LLM applications via multi-agent conversation. arXiv preprint arXiv:2308.08155, 2023; 1(1): 1-20.

YAO, Shunyu; ZHAO, Jeffrey; YU, Dian; DU, Nan; SHAFRAN, Izhak; NARASIMHAN, Karthik; CAO, Yuan. 2022. ReAct: Synergizing Reasoning and Acting in Language Models. In: *International Conference on Learning Representations (ICLR)*. arXiv:2210.03629.