White Paper - Completo

RELATÓRIO TÉCNICO COMPLETO

Retrieval-Augmented Generation (RAG) na Engenharia de Software

Revisão Sistemática Rápida com Análise Assistida por IA

1. Introdução

O avanço recente dos **Large Language Models (LLMs)** tem ampliado significativamente as possibilidades de automação e suporte a atividades cognitivas em diversos domínios, incluindo a Engenharia de Software. Esses modelos demonstram elevada capacidade de geração de texto, raciocínio contextual e suporte à tomada de decisão. No entanto, um dos principais desafios para sua aplicação em contextos industriais é a limitação de conhecimento específico de domínio e a tendência à geração de respostas imprecisas ou alucinações quando operam sem acesso a fontes confiáveis de informação (Lewis et al., 2020; Gao et al., 2023).

Nesse cenário, a abordagem conhecida como **Retrieval-Augmented Generation (RAG)** tem se consolidado como uma estratégia promissora para aumentar a confiabilidade e a utilidade prática dos LLMs. O RAG combina mecanismos de recuperação de informação com modelos generativos, permitindo que o sistema consulte bases de conhecimento externas, como documentação técnica, logs operacionais, código-fonte ou artefatos de desenvolvimento, antes de produzir uma resposta. Dessa forma, o modelo passa a fundamentar suas saídas em evidências recuperadas, reduzindo alucinações e aumentando a precisão das respostas (Lewis et al., 2020; Gao et al., 2023; Shuster et al., 2021).

Na Engenharia de Software, essa integração tem se mostrado particularmente relevante, uma vez que grande parte do conhecimento utilizado pelos desenvolvedores está distribuído em múltiplas fontes internas, como repositórios de código, histórico de incidentes, pipelines de CI/CD e documentação técnica. Sistemas baseados em RAG permitem que LLMs utilizem esse conhecimento organizacional para apoiar atividades como **revisão de código, geração de testes, diagnóstico de falhas, análise de logs e suporte operacional**, tornando-se ferramentas relevantes para aumento de produtividade e redução de esforço manual no desenvolvimento e manutenção de software.

Além disso, evidências recentes indicam que o RAG já vem sendo aplicado em diversos setores industriais, incluindo **computação em nuvem, telecomunicações, cibersegurança, sistemas corporativos e plataformas SaaS**, demonstrando seu potencial como tecnologia transversal para integração entre inteligência artificial generativa e conhecimento corporativo. Essas aplicações abrangem diferentes etapas do ciclo de

desenvolvimento de software, com destaque para atividades de desenvolvimento, testes, operações e resolução de incidentes.

Apesar desse crescimento, ainda existem lacunas importantes na literatura relacionadas à forma como organizações implementam arquiteturas de RAG em ambientes reais, incluindo decisões sobre escolha de modelos de linguagem, estratégias de embeddings, arquiteturas de recuperação e integração com pipelines de desenvolvimento. Compreender esses fatores é fundamental para orientar a adoção eficaz dessa tecnologia em contextos industriais.

Diante disso, este estudo busca responder à seguinte pergunta de pesquisa: **como o Retrieval-Augmented Generation está sendo aplicado na indústria de software?** Para explorar essa questão, investigamos: (i) quais indústrias estão adotando RAG, (ii) em quais etapas do ciclo de desenvolvimento de software essa abordagem é utilizada, (iii) quais modelos de linguagem e embeddings são empregados, e (iv) quais arquiteturas de implementação são adotadas. A partir da análise das evidências disponíveis na literatura e em relatos industriais, o objetivo é identificar padrões de adoção e fornecer recomendações práticas para o uso de RAG em ambientes de engenharia de software.

2. Metodologia

Inicialmente, foi definida a pergunta de pesquisa e, a partir dela, foi elaborada a string de busca, com base nos principais termos do problema investigado e seus sinônimos, visando garantir aderência ao escopo do estudo e cobertura adequada do tema.

Em seguida, foi conduzida uma busca automatizada na base IEEE, utilizando a string previamente definida. Essa busca retornou um total de 121 estudos.

Na etapa seguinte, foi realizada a leitura dos resumos dos artigos encontrados, com aplicação dos critérios de inclusão e exclusão estabelecidos no protocolo da revisão. Como resultado dessa triagem inicial, 20 estudos foram considerados aptos para a próxima fase.

Posteriormente,, os estudos selecionados passaram por leitura completa, permitindo avaliar sua aderência ao objetivo da pesquisa. Ao final dessa etapa, 15 estudos atenderam aos critérios estabelecidos e foram selecionados a partir da busca automatizada.

Foi aplicado o snowballing aos artigos encontrados no IEEE, por meio da busca de estudos que citaram os artigos selecionados, com o objetivo de complementar a busca automatizada e incluir trabalhos potencialmente relevantes não recuperados pela string ou pela base utilizada. Como resultado dessa etapa, 2 estudos adicionais foram identificados e incluídos no conjunto final de análise.

Com o intuito de ampliar a cobertura do estudo, foi realizada uma busca manual no Google Scholar, resultando na inclusão de 9 estudos adicionais considerados relevantes para o escopo da pesquisa.

Essa etapa teve como objetivo complementar a busca automatizada, ampliando a cobertura e reduzindo o risco de exclusão de trabalhos potencialmente relevantes que não foram recuperados pela string definida ou pela base selecionada.

Critérios de inclusão:

- Contexto industrial
- Integração com ciclo de desenvolvimento
- Escopo de RAG explícito
- Artigos entre 2021 e 2025
- Artigos na língua inglesa e portuguesa
- Artigos que não se encaixem nesse contexto serão desconsiderados

String de Busca:

("RAG" OR "retrieval augmented generation")AND("software engineering" OR "software development")AND("industry" OR "industrial" OR "industrial context" OR "industrial setting" OR "company" OR "companies" OR "enterprise*" OR firm* OR "organization*" OR organisational OR organizational OR "corporate" OR "in-house" OR commercial OR production OR workplace OR practitioner* OR professional* OR "real-world" OR "in practice" OR "study case")

3. Resultados

● Quais indústrias estão utilizando RAG ?

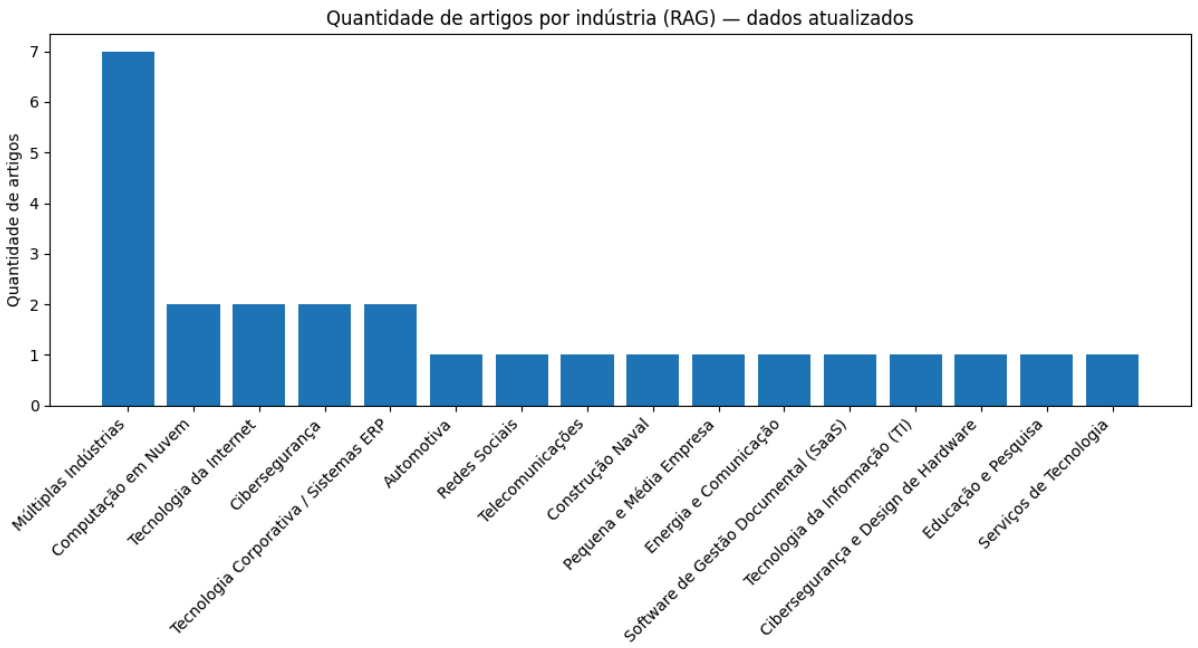
As indústrias identificadas nos estudos primários são apresentadas na tabela abaixo:

Indústria	Artigo(s)
Automotiva	Acceleration of Automotive Software Development by Retrieval Augmented Integration Test Script Generation
Computação em Nuvem	EagerLog: Active Learning Enhanced Retrieval Augmented Generation for Log-based Anomaly Detection; iKnow: an Intent-Guided Chatbot for Cloud Operations with Retrieval-Augmented Generation

Redes Sociais	A Deep Dive into Retrieval-Augmented Generation for Code Completion: Experience on WeChat
Tecnologia da Internet	Integrating RAG and LLM for Automated Code Review in Practice; LogSage: An LLM-Based Framework for CI/CD Failure Detection and Remediation with Industrial Validation
Telecomunicações	Developing a Llama-Based Chatbot for CI/CD Question Answering: A Case Study at Ericsson
Cibersegurança	An Empirical Evaluation of LLM-Based Approaches for Code Vulnerability Detection: RAG, SFT, and Dual-Agent Systems; Leveraging RAG and LLMs for Access Control Policy Extraction From User Stories in Agile Software Development
Construção Naval	Framework Design and implementation of the AI Aided Process Designing Platform for Shipbuilding Industry
Pequena e Média Empresa	An Agile Method for Implementing Retrieval-Augmented Generation Tools in Industrial SMEs
Tecnologia Corporativa / Sistemas ERP	Reinforcement Learning Integrated Agentic RAG for Software Test Cases Authoring; Agentic RAG for Software Testing with Hybrid Vector-Graph and Multi-Agent Orchestration
Energia e Comunicação	Large Language Model-Enhanced Test Case Generation
Software de Gestão Documental (SaaS).	Semantic Search in a Document Management System Using Retrieval Augmented Generation

Tecnologia da Informação (TI)	Deploying Large Language Models with Retrieval-Augmented Generation
Serviços de Tecnologia	Augmented Large Language Models for Software Engineering
Educação e Pesquisa	Seven Failure Points When Engineering a Retrieval Augmented Generation System
Cibersegurança e Design de Hardware	Retrieval-Augmented Code Generation: A Survey with Focus on Repository-Level Approaches
Múltiplas Indústrias	RAG-Based Code Intelligence for Real-Time Fault Diagnosis in Enterprise Systems; A Review on Retrieval-Augmented Generation: Architectures, Research Challenges, and Emerging Frontiers; A Systematic Review of Key Retrieval-Augmented Generation (RAG) Systems: Progress, Gaps, and Future Directions; A Systematic Literature Review of Retrieval-Augmented Generation: Techniques, Metrics, and Challenges; Retrieval-Augmented Generation in Industry: An Interview Study on Use Cases, Requirements, Challenges, and Evaluation; Retrieval-Augmented Text Generation: Methods, Challenges, and Applications; Advances in Retrieval-Augmented Generation (RAG) and Related Frameworks

A Figura **Quantidade de artigos por indústria(RAG)** apresenta a distribuição dos aspectos identificados nos estudos primários relacionados ao RAG.



Com base na tabela atualizada, os resultados mostram que o RAG já está sendo aplicado em múltiplos setores, com maior concentração em estudos classificados como Múltiplas Indústrias (7 estudos), indicando que parte relevante da literatura atual discute padrões, arquiteturas e desafios de forma transversal, independentemente do domínio. Entre os setores com maior evidência aplicada (2 estudos cada), destacam-se:

- **Computação em Nuvem**, com foco em observabilidade e operações: *EagerLog: Active Learning Enhanced Retrieval Augmented Generation for Log-based Anomaly Detection* (detecção de anomalias em logs) e *iKnow: an Intent-Guided Chatbot for Cloud Operations with Retrieval-Augmented Generation* (chatbot para OpsQA em cloud).

- **Tecnologia da Internet**, voltado a produtividade e confiabilidade em engenharia de software: *Integrating RAG and LLM for Automated Code Review in Practice* (revisão automática de código em ambiente corporativo) e *LogSage: An LLM-Based Framework for CI/CD Failure Detection and Remediation with Industrial Validation* (diagnóstico e remediação de falhas em CI/CD com validação industrial).

- **Cibersegurança**, aplicando RAG em tarefas críticas de segurança: *An Empirical Evaluation of LLM-Based Approaches for Code Vulnerability Detection: RAG, SFT, and Dual-Agent Systems* (detecção de vulnerabilidades com base MITRE/CWE) e **L***everaging RAG and LLMs for Access Control Policy Extraction From User Stories in Agile Software Development* (extração de políticas de controle de acesso a partir de histórias de usuário).

Nos demais domínios, os resultados indicam aplicações pontuais, mas bem definidas, reforçando a versatilidade do RAG:

- **Automotiva**: *Acceleration of Automotive Software Development by Retrieval Augmented Integration Test Script Generation*, voltado à geração de scripts de teste

de integração com suporte de manuais e conhecimento de workflow.

- **Redes Sociais**: *A Deep Dive into Retrieval-Augmented Generation for Code Completion: Experience on WeChat*, explorando RAG para completamento de código em repositório fechado.

- **Telecomunicações**: *Developing a Llama-Based Chatbot for CI/CD Question Answering: A Case Study at Ericsson*, aplicando chatbot RAG para dúvidas de CI/CD em ambiente CloudRAN.

- **Construção Naval**: *Framework Design and implementation of the AI Aided Process Designing Platform for Shipbuilding Industry*, usando RAG para gestão de conhecimento técnico e Q&A em engenharia naval.

- **Pequena e Média Empresa (PMEs)**: *An Agile Method for Implementing Retrieval-Augmented Generation Tools in Industrial SMEs*, propondo método ágil para implantação de RAG em contexto industrial com restrições.

- **Tecnologia Corporativa** / **Sistemas ERP**: *Agentic RAG for Software Testing with Hybrid Vector-Graph and Multi-Agent Orchestration* e *Reinforcement Learning Integrated Agentic RAG for Software Test Cases Authoring*, com foco na automação de artefatos de teste em cenários empresariais (ex.: SAP), incluindo orquestração multiagente e evolução via RL.

- **Energia e Comunicação**: *Large Language Model-Enhanced Test Case Generation*, aplicando RAG para geração de casos de teste a partir de requisitos e histórico de versões.

- **Software de Gestão Documental (SaaS)**: *Semantic Search in a Document Management System Using Retrieval Augmented Generation*, integrando RAG a um DMS para busca semântica e interação em linguagem natural.

- **Tecnologia da Informação (TI)**: *Deploying Large Language Models with Retrieval-Augmented Generation*, discutindo implantação e governança de soluções com RAG em ambiente corporativo.

- **Serviços de Tecnologia**: *Augmented Large Language Models for Software Engineering*, discutindo uso de LLMs aumentados (incluindo RAG) para apoiar a efetividade de profissionais de engenharia de software.

- **Educação e Pesquisa**: *Seven Failure Points When Engineering a Retrieval Augmented Generation System*, focado em pontos de falha e desafios de engenharia de sistemas RAG.

- **Cibersegurança e Design de Hardware**: *Retrieval-Augmented Code Generation: A Survey with Focus on Repository-Level Approaches*, consolidando evidências de geração de código aumentada por recuperação em nível de repositório.

A distribuição por indústria sugere três padrões relevantes: (i) uma base forte de estudos generalistas/multi-indústria que consolida arquiteturas, técnicas e desafios; (ii) crescimento de aplicações em Cloud/DevOps e Tecnologia da Internet, onde documentação operacional, logs e conhecimento histórico favorecem RAG; e (iii) adoção em segurança e testes, onde a necessidade de rastreabilidade, redução de alucinação e reutilização de conhecimento histórico torna o RAG especialmente atrativo.

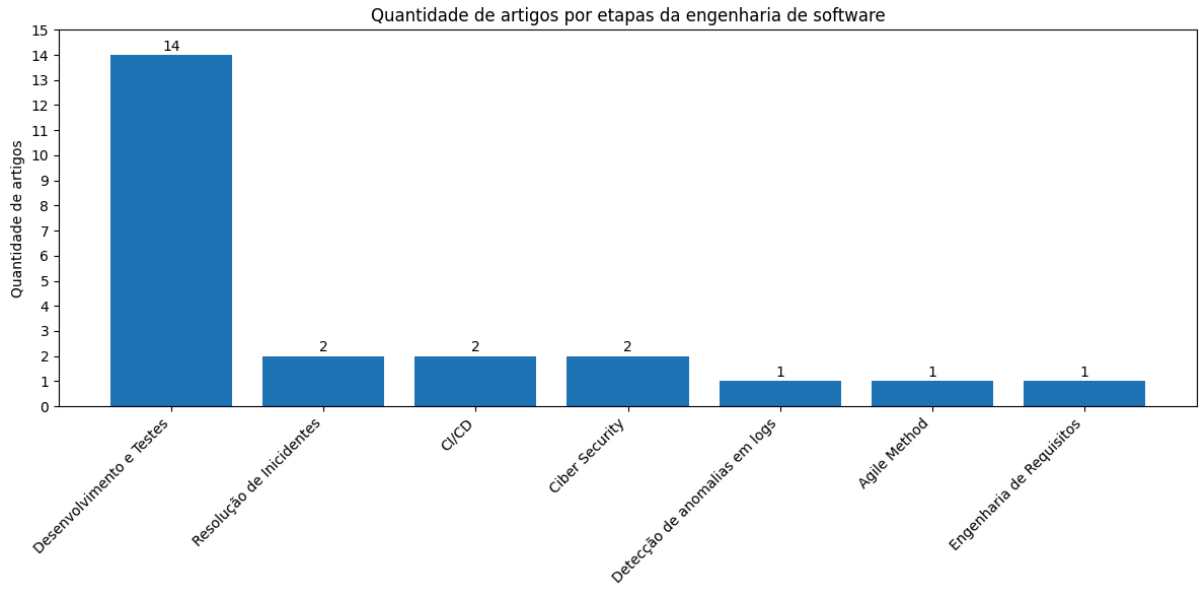
● Em quais etapas do desenvolvimento de software o RAG está sendo utilizado?

As etapas da Engenharia de Software ao qual utilizam RAG podem ser consideradas conforme a tabela abaixo:

Engenharia de software	Artigos
Detecção de anomalias em logs	EagerLog: Active Learning Enhanced Retrieval Augmented Generation for Log-based Anomaly Detection
Resolução de Incidentes	iKnow: an Intent-Guided Chatbot for Cloud Operations with Retrieval-Augmented Generation; RAG-Based Code Intelligence for Real-Time Fault Diagnosis in Enterprise Systems
CI/CD	LogSage: An LLM-Based Framework for CI/CD Failure Detection and Remediation with Industrial Validation; Developing a Llama-Based Chatbot for CI/CD Question Answering: A Case Study at Ericsson
Ciber Security	An Empirical Evaluation of LLM-Based Approaches for Code Vulnerability Detection: RAG, SFT, and Dual-Agent Systems; Leveraging RAG and LLMs for Access Control Policy Extraction From User Stories in Agile Software Development
Agile Method	An Agile Method for Implementing Retrieval-Augmented Generation Tools in Industrial SMEs

Desenvolvimento e Testes	Deploying Large Language Models with Retrieval-Augmented Generation; A Systematic Review of Key Retrieval-Augmented Generation (RAG) Systems: Progress, Gaps, and Future Directions; Retrieval-Augmented Code Generation: A Survey with Focus on Repository-Level Approaches; Seven Failure Points When Engineering a Retrieval Augmented Generation System; Augmented Large Language Models for Software Engineering; Acceleration of Automotive Software Development by Retrieval Augmented Integration Test Script Generation; Reinforcement Learning Integrated Agentic RAG for Software Test Cases Authoring; Agentic RAG for Software Testing with Hybrid Vector-Graph and Multi-Agent Orchestration; Large Language Model-Enhanced Test Case Generation; A Deep Dive into Retrieval-Augmented Generation for Code Completion: Experience on WeChat; Integrating RAG and LLM for Automated Code Review in Practice; Semantic Search in a Document Management System Using Retrieval Augmented Generation; Retrieval-Augmented Text Generation: Methods, Challenges, and Applications; Retrieval-Augmented Generation in Industry: An Interview Study on Use Cases, Requirements, Challenges, and Evaluation
Engenharia de Requisitos	Advances in Retrieval-Augmented Generation (RAG) and Related Frameworks

A Figura **Quantidade de artigos por etapas da Engenharia de Software** apresenta a distribuição dos aspectos identificados nos estudos primários relacionados ao RAG.



A etapa Desenvolvimento e Testes (14 estudos) concentra a maior parte das evidências e reúne um conjunto amplo de problemas-alvo que justificam o uso de RAG em contextos industriais. Nessa etapa, o RAG é empregado para (i) aumentar produtividade em atividades de desenvolvimento, como completamento de código e apoio ao trabalho em repositórios fechados, conforme *A Deep Dive into Retrieval-Augmented Generation for Code Completion: Experience on WeChat*; (ii) melhorar qualidade durante o desenvolvimento, fornecendo contexto para revisão de código e reduzindo sugestões inválidas, como em *Integrating RAG and LLM for Automated Code Review in Practice*; (iii) automatizar e aprimorar a geração de testes e artefatos de qualidade, reduzindo esforço manual, aumentando cobertura e promovendo rastreabilidade, evidenciado por *Acceleration of Automotive Software Development by Retrieval Augmented Integration Test Script Generation, Agentic RAG for Software Testing with Hybrid Vector-Graph and Multi-Agent Orchestration, Reinforcement Learning Integrated Agentic RAG for Software Test Cases Authoring* e *Large Language Model-Enhanced Test Case Generation*; e (iv) endereçar desafios de engenharia e operacionalização do RAG em produção, como governança, falhas do pipeline, trade-offs de latência versus qualidade e avaliação contínua, discutidos em *Deploying Large Language Models with Retrieval-Augmented Generation, Seven Failure Points When Engineering a Retrieval Augmented Generation System, Retrieval-Augmented Generation in Industry: An Interview Study on Use Cases, Requirements, Challenges, and Evaluation, Retrieval-Augmented Text Generation: Methods, Challenges, and Applications, A Systematic Review of Key Retrieval-Augmented Generation (RAG) Systems: Progress, Gaps, and Future Directions, Retrieval-Augmented Code Generation: A Survey with Focus on Repository-Level Approaches* e *Augmented Large Language Models for Software Engineering*. Além disso, *Semantic Search in a Document Management System Using Retrieval Augmented Generation* integra esse bloco ao evidenciar o RAG como apoio a acesso de conhecimento e busca semântica em artefatos/documentos usados no desenvolvimento.

Na sequência, três etapas aparecem com volume equivalente de evidências (2 estudos cada). Em Resolução de Incidentes (2 estudos), o RAG é aplicado para reduzir tempo de diagnóstico e recuperação (MTTR), apoiar troubleshooting e aumentar confiança/explicabilidade por meio de evidências recuperadas de documentação, histórico e dados operacionais, conforme *iKnow: an Intent-Guided Chatbot for Cloud Operations with Retrieval-Augmented Generation* e *RAG-Based Code Intelligence for Real-Time Fault Diagnosis in Enterprise Systems*.

Em CI/CD (2 estudos), o RAG é utilizado para enfrentar problemas de falhas recorrentes em pipelines e necessidade de remediação e orientação operacional, reduzindo esforço manual na interpretação de logs e no acesso a conhecimento histórico frequentemente disperso ou não estruturado. Esse objetivo aparece em *LogSage: An LLM-Based Framework for CI/CD Failure Detection and Remediation with Industrial Validation* e *Developing a Llama-Based Chatbot for CI/CD Question Answering: A Case Study at Ericsson*.

Em Ciber Security (2 estudos), o RAG é empregado para tarefas de alto impacto, nas quais é essencial fundamentar respostas em conhecimento específico e confiável, reduzindo alucinações e aumentando precisão. Os problemas-alvo incluem detecção de vulnerabilidades e extração/validação de políticas de controle de acesso a partir de artefatos

textuais (histórias de usuário), conforme *An Empirical Evaluation of LLM-Based Approaches for Code Vulnerability Detection: RAG, SFT, and Dual-Agent Systems* e *Leveraging RAG and LLMs for Access Control Policy Extraction From User Stories in Agile Software Development*.

Por fim, aparecem etapas com 1 estudo cada. Em Detecção de anomalias em logs (1 estudo), o foco é detectar anomalias com poucos dados rotulados e manter uma base de conhecimento útil mesmo com evolução do sistema, como em *EagerLog: Active Learning Enhanced Retrieval Augmented Generation for Log-based Anomaly Detection*. Em Agile Method (1 estudo), o problema central é viabilizar implantação de RAG em PMEs sob restrições de custo, tempo e equipe, representado por *An Agile Method for Implementing Retrieval-Augmented Generation Tools in Industrial SMEs*. Em Engenharia de Requisitos (1 estudo), o estudo analisado contribui ao discutir avanços e desafios associados ao uso de RAG em ambientes empresariais e dados proprietários (incluindo privacidade e latência), como em *Advances in Retrieval-Augmented Generation (RAG) and Related Frameworks*.

Em síntese, a distribuição evidencia que o RAG tem sido adotado principalmente para resolver problemas de integração com conhecimento corporativo e aumento de confiabilidade, com maior concentração em Desenvolvimento e Testes, e aplicações relevantes também em Resolução de Incidentes, CI/CD e Ciber Security.

- **Quais modelos de LLM estão sendo utilizados em conjunto com RAG ?**

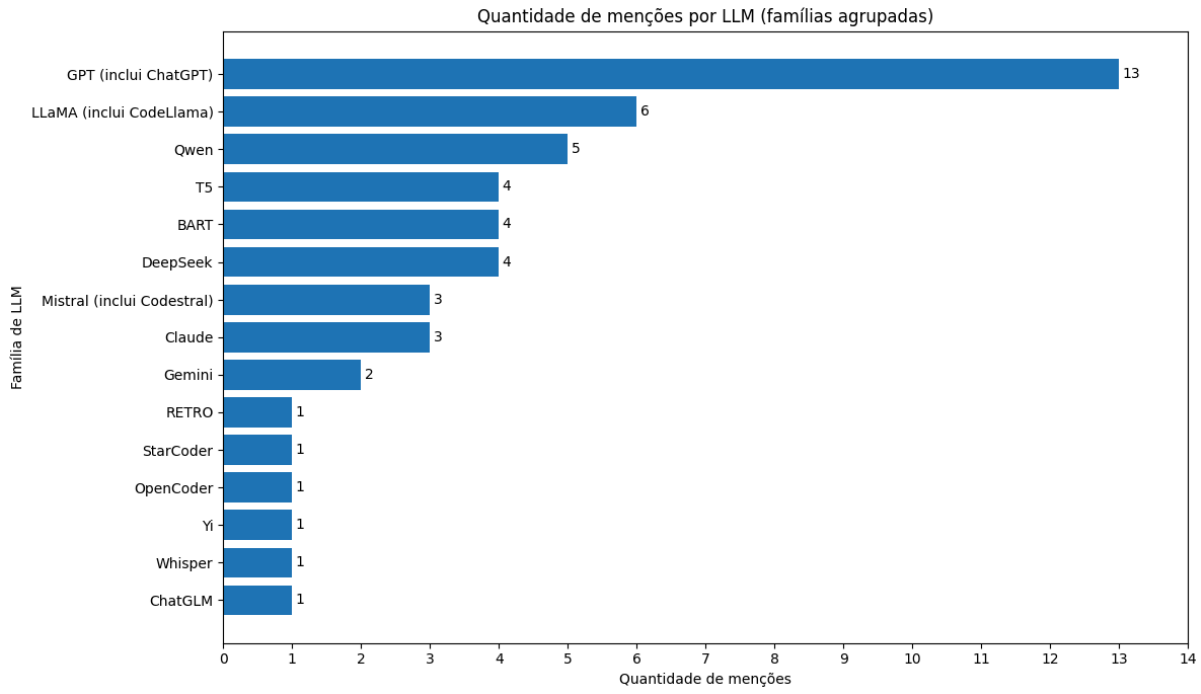
As LLM associadas ao RAG podem ser considerados conforme a tabela abaixo:

Artigos	LLMs citadas
EagerLog: Active Learning Enhanced Retrieval Augmented Generation for Log-based Anomaly Detection; *iKnow: an Intent-Guided Chatbot for Cloud Operations with Retrieval-Augmented Generation*	Qwen; Qwen-2.5-32B-instruct
A Deep Dive into Retrieval-Augmented Generation for Code Completion: Experience on WeChat	Qwen-Coder; Deepseek-Coder; CodeLlama; Yi-Coder; OpenCoder; Codestral

Integrating RAG and LLM for Automated Code Review in Practice; *LogSage: An LLM-Based Framework for CI/CD Failure Detection and Remediation with Industrial Validation*	GPT o1; DeepSeek-R1; Claude 3.7 Sonnet; GPT-4o; Claude-3.7-Sonnet; Deepseek V3
Developing a Llama-Based Chatbot for CI/CD Question Answering: A Case Study at Ericsson	Llama2-chat
An Empirical Evaluation of LLM-Based Approaches for Code Vulnerability Detection: RAG, SFT, and Dual-Agent Systems; *Leveraging RAG and LLMs for Access Control Policy Extraction From User Stories in Agile Software Development*	LLaMA 3.2-3B;
Framework Design and implementation of the AI Aided Process Designing Platform for Shipbuilding Industry	Qwen2-72B-instruction; Qwen2-7B-instruction; Chatglm-4-9B-chat
An Agile Method for Implementing Retrieval-Augmented Generation Tools in Industrial SMEs	gpt-3.5-turbo
Reinforcement Learning Integrated Agentic RAG for Software Test Cases Authoring; *Agentic RAG for Software Testing with Hybrid Vector-Graph and Multi-Agent Orchestration*	Mistral 7B; Gemini Pro
Semantic Search in a Document Management System Using Retrieval Augmented Generation	ChatGPT
Seven Failure Points When Engineering a Retrieval Augmented Generation System	GPT-4; GPT-3.5; Whisper
Retrieval-Augmented Code Generation: A Survey with Focus on Repository-Level Approaches	LLaMA; GPT; Claude; DeepSeek-Coder; StarCoder

RAG-Based Code Intelligence for Real-Time Fault Diagnosis in Enterprise Systems; *A Review on Retrieval-Augmented Generation: Architectures, Research Challenges, and Emerging Frontiers*; *A Systematic Review of Key Retrieval-Augmented Generation (RAG) Systems: Progress, Gaps, and Future Directions*; *A Systematic Literature Review of Retrieval-Augmented Generation: Techniques, Metrics, and Challenges*; *Retrieval-Augmented Generation in Industry: An Interview Study on Use Cases, Requirements, Challenges, and Evaluation*; *Retrieval-Augmented Text Generation: Methods, Challenges, and Applications*; *Advances in Retrieval-Augmented Generation (RAG) and Related Frameworks*	Mistral-7B; GPT-4; BART; T5; GPT (3+); BART; T5; GPT-4; LLaMA; BART; T5; GPT (diversos); LLaMA; RETRO; ChatGPT; Gemini; T5; BART; GPT;

A Figura **Quantidade de menções por LLM** apresenta a distribuição dos aspectos identificados nos estudos primários relacionados ao RAG.



Família GPT (ChatGPT, GPT-3.5, GPT-4, GPT-4o, GPT o1)

A família GPT aparece com alta frequência por dois motivos principais: capacidade de seguir instruções e gerar respostas consistentes quando ancoradas em evidências recuperadas, e maturidade de ecossistema (ferramentas, integração e padronização de pipelines RAG). Em aplicações industriais, isso se traduz em uso direto como motor de resposta em sistemas de gestão do conhecimento e suporte operacional, por exemplo, em um DMS com busca semântica baseado em RAG usando ChatGPT como LLM, visando consultas em linguagem natural e iteração sobre resultados ("follow-up queries") (*Semantic Search in a Document Management System Using Retrieval Augmented Generation*).

Além disso, na engenharia de sistemas RAG, GPT é frequentemente escolhido para identificar falhas típicas (como ruído de recuperação, reescrita de consulta e erros de consolidação), como discutido em *Seven Failure Points When Engineering a Retrieval Augmented Generation System* (GPT-4 e GPT-3.5). Em tarefas mais críticas e de "produção", a escolha por GPT também aparece como parte do conjunto de modelos avaliados (ex.: GPT-4o no LogSage) para diagnosticar e sugerir remediação em falhas de CI/CD, onde o objetivo é reduzir MTTR e produzir guias acionáveis (*LogSage: An LLM-Based Framework for CI/CD Failure Detection and Remediation***)**.

Por fim, em cenários de diagnóstico de incidentes em tempo real, a combinação de GPT-4 com RAG é usada como motor de raciocínio para correlacionar logs, traços e evidências históricas, priorizando aplicabilidade e confiança do desenvolvedor (*RAG-Based Code Intelligence for Real-Time Fault Diagnosis in Enterprise Systems*).

Família LLaMA (Llama2-chat, LLaMA 3.x) e derivados de código (CodeLlama)

A família LLaMA se destaca quando a prioridade é implantação controlada (on-premises), custo e governança. Isso aparece claramente no estudo de caso de chatbot para CI/CD na Ericsson, que usa Llama2-chat com RAG e avalia componentes como reescrita de consulta, busca híbrida e compressão de contexto, ressaltando inclusive o custo de trocar de LLM por diferenças de estilo de prompt (*Developing a Llama-Based Chatbot for CI/CD Question Answering: A Case Study at Ericsson*).

Em cibersegurança, o uso de LLaMA 3.2-3B aparece como modelo de base para comparar abordagens (RAG, fine-tuning e dual-agent) em detecção de vulnerabilidades, onde a vantagem do RAG é reduzir alucinações ao ancorar no CWE/MITRE (*An Empirical Evaluation of LLM-Based Approaches for Code Vulnerability Detection: RAG, SFT, and Dual-Agent Systems*).

Já em desenvolvimento, CodeLlama surge como alternativa específica para tarefas de código em estudo de completamento em repositório fechado (WeChat), reforçando a tendência de "modelos especializados em código" competirem com modelos gerais quando o domínio é programação (*A Deep Dive into Retrieval-Augmented Generation for Code Completion: Experience on WeChat*).

Família Qwen (Qwen, Qwen-2.5-32B-instruct, Qwen2-72B/7B)

A família Qwen aparece fortemente associada a operações e ambientes de nuvem, sugerindo preferência por modelos que equilibram capacidade e custo/implantação em

pipelines de suporte. Em detecção de anomalias em logs, Qwen é usado como o LLM que recebe evidências recuperadas do banco vetorial e produz classificação e explicação (*EagerLog: Active Learning Enhanced Retrieval Augmented Generation for Log-based Anomaly Detection*).

Em operações de nuvem, o iKnow utiliza Qwen-2.5-32B-instruct, com uma arquitetura sofisticada de RAG (múltiplos VecDB por versão, intent detection, query rewriting e reranking), refletindo uma escolha orientada a robustez e governança do conhecimento (*iKnow: an Intent-Guided Chatbot for Cloud Operations with Retrieval-Augmented Generation*).

Já em indústria tradicional (construção naval), Qwen2-72B e Qwen2-7B aparecem como opções para uma plataforma de gestão de conhecimento técnico baseada em RAG, indicando que modelos open-weight/regionais também entram como escolha por viabilidade e integração em sistemas corporativos (*Framework Design and implementation of the AI Aided Process Designing Platform for Shipbuilding Industry*). No desenvolvimento, Qwen-Coder aparece como modelo avaliado para completamento de código em repositório fechado (*A Deep Dive into RAG for Code Completion: Experience on WeChat*).

Modelos "alternativos" usados por comparação, custo, ou ajuste ao domínio (Claude, Mistral, Gemini, DeepSeek)

Alguns trabalhos não usam apenas uma única família: eles comparam múltiplos LLMs para medir desempenho ou equilibrar custo. Em CI/CD, LogSage avalia GPT-4o, Claude-3.7-Sonnet e DeepSeek V3, mostrando que aplicações críticas testam diferentes LLMs para maximizar precisão e reduzir tempo de resolução (LogSage: An LLM-Based Framework for CI/CD Failure Detection and Remediation with Industrial Validation).

Em revisão de código, modelos como Claude 3.7 Sonnet e DeepSeek-R1, além de GPT o1, aparecem como motores avaliados para gerar comentários úteis e reduzir sugestões inválidas por filtragem (*Integrating RAG and LLM for Automated Code Review in Practice*).

No campo de testes com RAG agêntico, aparecem Mistral 7B e Gemini Pro como modelos integrados a uma arquitetura vetor-grafo e orquestração multiagente (*Agentic RAG for Software Testing with Hybrid Vector-Graph and Multi-Agent Orchestration*).

Em completamento de código, modelos DeepSeek-Coder, Yi-Coder, Codestral e OpenCoder reforçam a tendência de comparar modelos especializados em código (*A Deep Dive into Retrieval-Augmented Generation for Code Completion: Experience on WeChat*).

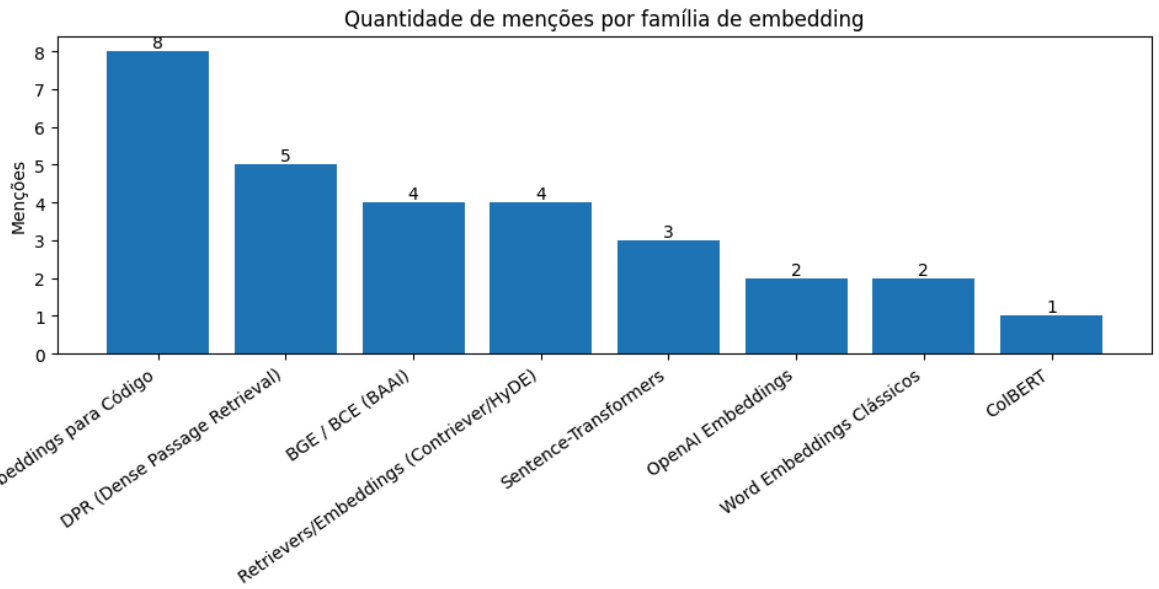
● Quais modelos de embeddings estão sendo utilizados com RAG?

Os embeddings associados ao RAG podem ser considerados conforme a tabela abaixo

Artigos	Embeddings citados
EagerLog: Active Learning Enhanced Retrieval Augmented Generation for Log-based Anomaly Detection	DPR encoder (Dense Passage Retrieval)
iKnow: an Intent-Guided Chatbot for Cloud Operations with Retrieval-Augmented Generation	BGE-m3 *(e também aparece bce-reranker-base_v1 como reranker)*
A Deep Dive into Retrieval-Augmented Generation for Code Completion: Experience on WeChat	CodeBERT; UniXcoder; CoCoSoDa; GTE-Qwen
LogSage: An LLM-Based Framework for CI/CD Failure Detection and Remediation with Industrial Validation	BGE (BAAI General Embedding)
Developing a Llama-Based Chatbot for CI/CD Question Answering: A Case Study at Ericsson	BAAI/bge-base-en
Leveraging RAG and LLMs for Access Control Policy Extraction From User Stories in Agile Software Development	sentence-transformers/all-distilroberta-v1
Deploying Large Language Models with Retrieval-Augmented Generation	OpenAIEmbeddings
RAG-Based Code Intelligence for Real-Time Fault Diagnosis in Enterprise Systems	text-embedding-3-large

An Agile Method for Implementing Retrieval-Augmented Generation Tools in Industrial SMEs	all-MiniLM-L6-v2
Agentic RAG for Software Testing with Hybrid Vector-Graph and Multi-Agent Orchestration	Sentence Transformer *(família/técnica citada no artigo)*
A Review on Retrieval-Augmented Generation: Architectures, Research Challenges, and Emerging Frontiers	DPR; ColBERT *(citados como mecanismos/modelos de embeddings na discussão)*
A Systematic Review of Key Retrieval-Augmented Generation (RAG) Systems: Progress, Gaps, and Future Directions	DPR; Contriever; HyDE
A Systematic Literature Review of Retrieval-Augmented Generation: Techniques, Metrics, and Challenges	DPR; Contriever; HyDE
Retrieval-Augmented Code Generation: A Survey with Focus on Repository-Level Approaches	UniXcoder; CodeT5; Voyage-Code; CodeBERT
Retrieval-Augmented Text Generation: Methods, Challenges, and Applications	DPR; Word2Vec; GloVe

A Figura **Quantidade de menções por família de Embedding** apresenta a distribuição dos aspectos identificados nos estudos primários relacionados ao RAG.



A família BGE (BAAI) foi a mais recorrente na amostra. Ela aparece como base de vetorização em cenários industriais que exigem alta precisão de recuperação e, frequentemente, mecanismos adicionais de refinamento. Isso é observado em *iKnow: an Intent-Guided Chatbot for Cloud Operations with Retrieval-Augmented Generation*, que utiliza BGE-m3 para vetorização e incorpora um reranker (bce-reranker-base_v1) para reordenar evidências, fortalecendo a recuperação antes da geração. A mesma tendência de embeddings robustos para ambientes operacionais também aparece em *LogSage: An LLM-Based Framework for CI/CD Failure Detection and Remediation with Industrial Validation*, que adota BGE (BAAI General Embedding) em seu pipeline de recuperação. Já no contexto de perguntas e respostas sobre CI/CD em uma organização de telecom, Developing a Llama-Based Chatbot for CI/CD Question Answering: A Case Study at Ericsson emprega BAAI/bge-base-en, reforçando o uso da família BGE como escolha recorrente em cenários de suporte a engenharia e operação, onde o custo do erro pode ser alto.

Em segundo lugar, a família Sentence-Transformers aparece com frequência como opção pragmática e custo-efetiva, especialmente em cenários de implementação aplicada e com foco em viabilidade. Em *An Agile Method for Implementing Retrieval-Augmented Generation Tools in Industrial SMEs (EASI-RAG)*, o embedding citado é all-MiniLM-L6-v2, típico de pipelines leves para implantação rápida. Em *Leveraging RAG and LLMs for Access Control Policy Extraction From User Stories in Agile Software Development*, o modelo sentence-transformers/all-distilroberta-v1 é utilizado para representar semanticamente histórias de usuário e especificações, alinhando a recuperação ao domínio de requisitos e segurança. A família também aparece de forma mais genérica em *Agentic RAG for Software Testing with Hybrid Vector-Graph and Multi-Agent Orchestration*, indicada como Sentence Transformer, sugerindo adoção por facilidade de integração e por ser amplamente suportada no ecossistema de vetores/IR.

A terceira linha mais presente envolve embeddings do ecossistema OpenAI, usados principalmente quando o pipeline já está acoplado ao uso de modelos da OpenAI. Em *RAG-Based Code Intelligence for Real-Time Fault Diagnosis in Enterprise Systems*, o

embedding explicitado é text-embedding-3-large, reforçando a estratégia de alinhar o modelo de embeddings ao provedor/stack de geração. De forma semelhante, *Deploying Large Language Models with Retrieval-Augmented Generation* relata o uso de OpenAIEmbeddings (via LangChain) em conjunto com GPT-4 Turbo, evidenciando uma escolha guiada por compatibilidade e padronização do pipeline.

Além desses grupos dominantes, surgem embeddings mais “de nicho”, fortemente dependentes do tipo de dado e do objetivo do RAG. No contexto de observabilidade e logs, *EagerLog: Active Learning Enhanced Retrieval Augmented Generation for Log-based Anomaly Detection* utiliza um DPR encoder (Dense Passage Retrieval) para transformar sequências de logs em vetores, refletindo uma escolha clássica de retrieval denso aplicada ao domínio de logs. No caso de RAG aplicado a repositórios fechados e tarefas centradas em código, *A Deep Dive into Retrieval-Augmented Generation for Code Completion: Experience on WeChat* explicita o uso de embeddings especializados para código como CodeBERT, UniXcoder, CoCoSoDa e GTE-Qwen além de técnicas híbridas com BM25, mostrando que, quando o artefato principal é código, a literatura tende a preferir embeddings treinados ou avaliados diretamente para similaridade de código.

- # Quais arquiteturas de implementação de RAG são adotados?

Pipeline RAG padrão

Uma parte dos estudos utiliza o RAG padrão padrão “clássico” de RAG: documentos são pré-processados, segmentados em chunks, vetorizados e armazenados em um banco vetorial, e a consulta do usuário aciona a recuperação do contexto para geração pelo LLM. Esse padrão aparece claramente em *An Empirical Evaluation of LLM-Based Approaches for Code Vulnerability Detection: RAG, SFT, and Dual-Agent Systems*, que detalha ingestão, chunking, embeddings e armazenamento em Chroma, seguido de recuperação e geração. Um arranjo similar é descrito em *Deploying Large Language Models with Retrieval-Augmented Generation*, com um fluxo de orquestração que prepara documentos, indexa e usa recuperação por similaridade para alimentar o LLM.

RAG por prefixo de código

Em tarefas de geração de scripts/código em ambientes especializados, surge uma arquitetura que combina RAG com prefixos de código para controlar a forma e aumentar confiabilidade do resultado. Em *Acceleration of Automotive Software Development by Retrieval Augmented Integration Test Script Generation*, o modelo recebe (i) um vector store com documentação (manuais + conhecimento de workflow) e (ii) um prefixo de código inicializado conforme o hardware, e a geração é restrita a completar o script a partir desse prefixo. Essa arquitetura é relevante por mostrar um padrão industrial: usar RAG para grounding e usar “estrutura fixa” para reduzir variabilidade/alucinação.

RAG com base de conhecimento dinâmica (Active Learning / KB evolutiva)

Alguns estudos tratam a base de conhecimento como um ativo que precisa evoluir para evitar obsolescência e ruído. Em *EagerLog: Active Learning Enhanced Retrieval Augmented Generation for Log-based Anomaly Detection*, o RAG inicia com uma base pequena e confiável e incorpora um loop de aprendizado ativo, em que logs com baixa incerteza podem ser incorporados automaticamente e casos incertos são enviados para rotulagem humana. Aqui, a arquitetura se destaca por tornar o RAG um sistema auto-atualizável, adequado para cenários em que logs mudam rapidamente.

RAG guiado por intenção (Intent Detection + Query Rewriting + KB versionada)

Na operação de nuvem e suporte, aparece uma arquitetura em que a qualidade da resposta depende menos do LLM "em si" e mais de entender intenção e reformular consulta. Em *iKnow: an Intent-Guided Chatbot for Cloud Operations with Retrieval-Augmented Generation*, a arquitetura inclui: (i) criação de múltiplos bancos vetoriais por versão de documentação, (ii) detecção de intenção para classificar o tipo de consulta, (iii) extração de metadados (ex.: aplicação e versão), (iv) reescrita de consulta guiada por intenção e (v) geração baseada no contexto recuperado. Esse padrão mostra como RAG industrial evolui para lidar com consultas incompletas e com documentos versionados.

Recuperação em dois estágios (Retrieve + Rerank) e controle de qualidade do contexto

Diversos trabalhos incorporam um segundo estágio para filtrar ruído: primeiro recuperam um conjunto maior (alta revocação) e depois aplicam um re-ranqueador (alta precisão). Em iKnow, o retrieval inicial é seguido por reranking via cross-encoder antes de selecionar o top-k final. Em *LogSage: An LLM-Based Framework for CI/CD Failure Detection and Remediation with Industrial Validation*, além de múltiplas estratégias de recuperação, há reclassificação/reranking e cuidados para estabilizar respostas. Uma ideia similar de pós-processamento de evidências aparece também em *Leveraging RAG and LLMs for Access Control Policy Extraction From User Stories in Agile Software Development*, que combina recuperação e ranqueamento antes da etapa generativa.

RAG híbrido (lexical + semântico) com fusão e estratégias de ensemble

Um padrão recorrente é a adoção de busca híbrida para equilibrar vocabulário específico e similaridade semântica. Em *A Deep Dive into Retrieval-Augmented Generation for Code Completion: Experience on WeChat*, o estudo mostra dois RAGs (por identificadores e por similaridade) e destaca que a melhor configuração combina recuperação lexical (BM25) e recuperação semântica. Em *Developing a Llama-Based Chatbot for CI/CD Question Answering: A Case Study at Ericsson*, a escolha mais efetiva envolve um recuperador "ensemble" combinando BM25 e embeddings. Em *Large Language Model-Enhanced Test Case Generation*, a arquitetura combina TF-IDF e embeddings BERT com fusão por Reciprocal Rank Fusion (RRF), reforçando que a fusão de sinais é uma decisão arquitetural central quando o sistema precisa lidar com versões, mudanças e rastros históricos.

RAG multi-rota e multi-fonte

Em cenários industriais de alta variabilidade (por exemplo, falhas de CI/CD), surge um padrão de engenharia de recuperação mais “robusto”: recuperar por várias rotas e consolidar. O exemplo mais claro é LogSage, que descreve recuperação multi-rota combinando diferentes bases, granularidades e métricas (lexical e vetorial), seguida de reranking e mecanismos de estabilização do output (ex.: tratamento de URLs). Esse padrão é típico de produção, onde o conhecimento está espalhado e a arquitetura precisa garantir recall sem sacrificar precisão.

RAG com grafos e Agentic RAG

Para tarefas com dependências estruturais (código, chamadas entre arquivos, rastreabilidade), aparecem arquiteturas baseadas em grafos e/ou multi-agentes. Em *Integrating RAG and LLM for Automated Code Review in Practice*, há recuperação por cadeia de chamadas a partir de uma base em grafo de código, o que permite contexto entre arquivos e impacto da mudança. Já em *Agentic RAG for Software Testing with Hybrid Vector-Graph and Multi-Agent Orchestration*, a arquitetura combina banco vetorial + banco em grafo e uma camada de orquestração multi-agente, com motor de contextualização e rastreabilidade. Em *Reinforcement Learning Integrated Agentic RAG for Software Test Cases Authoring*, esse padrão é estendido com aprendizado por reforço para permitir melhoria contínua baseada em feedback, reforçando uma tendência de RAG como sistema “aprendente”, e não apenas recuperador.

5. Conclusão Geral

As evidências analisadas indicam que o RAG já ultrapassou o estágio experimental e começa a ser utilizado em ambientes industriais, especialmente em atividades relacionadas a desenvolvimento, operações e segurança de software.

Seu principal valor está em permitir que LLMs utilizem conhecimento corporativo e dados proprietários, tornando possível aplicar inteligência artificial generativa em contextos organizacionais complexos.

No entanto, a eficácia dessas soluções depende da qualidade do pipeline de recuperação, da curadoria do conhecimento e da integração adequada com os fluxos de desenvolvimento de software.

Artigos

Referências

Gao, L., Ma, X., Lin, J., & Callan, J. (2024). **Retrieval-Augmented Generation for Large Language Models: A Survey.** arXiv preprint arXiv:2312.10997.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W., Rocktäschel, T., Riedel, S., & Kiela, D. (2020). **Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks.** Advances in Neural Information Processing Systems (NeurIPS).

Shuster, K., Poff, S., Chen, M., Kiela, D., & Weston, J. (2021). **Retrieval Augmentation Reduces Hallucination in Conversation.** Findings of the Association for Computational Linguistics (ACL Findings).

Acceleration of Automotive Software Development by Retrieval Augmented Integration Test Script Generation — https://drive.google.com/file/d/19tUWOW9hxlgMREZg1I1m3ELj8cLq1Gzt/view?usp=drive_link

EagerLog: Active Learning Enhanced Retrieval Augmented Generation for Log-based Anomaly Detection — https://drive.google.com/file/d/1phvI2RCUHumhobHsL0QJLCFKzHzc3PTo/view?usp=drive_link

iKnow: an Intent-Guided Chatbot for Cloud Operations with Retrieval-Augmented Generation — https://drive.google.com/file/d/1Vm0JzGrrjZlAprkgnpoRrQnO1WLP2luS/view?usp=drive_link

A Deep Dive into Retrieval-Augmented Generation for Code Completion: Experience on WeChat — https://drive.google.com/file/d/1uGFUQpznOioUqIDsGXBa1T6Fc4WSbZhL/view?usp=drive_link

Integrating RAG and LLM for Automated Code Review in Practice — https://drive.google.com/file/d/1huGJihFp0Fh9ubgIchwIFokDbhjdIeaF/view?usp=drive_link

LogSage: An LLM-Based Framework for CI/CD Failure Detection and Remediation with Industrial Validation — https://drive.google.com/file/d/1shhCgREekRnGM95IhQq5c5JYqHLW4UmU/view?usp=drive_link

Developing a Llama-Based Chatbot for CI/CD Question Answering: A Case Study at Ericsson — https://drive.google.com/file/d/18DHdqc1W0qDmUUDeCHjsaSbVk5eGroAb/view?usp=drive_link

An Empirical Evaluation of LLM-Based Approaches for Code Vulnerability Detection: RAG, SFT, and Dual-Agent Systems — https://drive.google.com/file/d/17LFc-Br1tpKyDIL2NZPJQwtdYahyLlqZ/view?usp=drive_link

Leveraging RAG and LLMs for Access Control Policy Extraction From User Stories in Agile Software Development —

https://drive.google.com/file/d/1SZUEE1zwY7i-iALDFexP1KYwEVCTL-t4/view?usp=drive_link

Framework Design and implementation of the AI Aided Process Designing Platform for Shipbuilding Industry — https://drive.google.com/file/d/1mG6r-QwWLvaEAs7IPjjTzK22QKKlysA3/view?usp=drive_link

An Agile Method for Implementing Retrieval-Augmented Generation Tools in Industrial SMEs — https://drive.google.com/file/d/1XQtySoac84HYsYXu1qWPQUT_IEJYsXIN/view?usp=drive_link

Reinforcement Learning Integrated Agentic RAG for Software Test Cases Authoring — https://drive.google.com/file/d/1chT9E2g0lJKWlf8eAI7uRSUTSYyQiqrc/view?usp=drive_link

Agentic RAG for Software Testing with Hybrid Vector-Graph and Multi-Agent Orchestration — https://drive.google.com/file/d/1ZJiP9TngI5FQ6KXJXfWvsSa3cTk_jJOw/view?usp=drive_link

Large Language Model-Enhanced Test Case Generation — https://drive.google.com/file/d/1H5MqpVH-cSKbyI6JE3JM_fnJOd_ZQltb/view?usp=drive_link

RAG-Based Code Intelligence for Real-Time Fault Diagnosis in Enterprise Systems — https://drive.google.com/file/d/1QzMffvN3FnXoC6norpkFgKOsqTOYWDA2/view?usp=drive_link

Semantic Search in a Document Management System Using Retrieval Augmented Generation — https://drive.google.com/file/d/1zIlbE2azyo9k947Y7XcdEsVVpfmVOCFR/view?usp=drive_link

A Review on Retrieval-Augmented Generation: Architectures, Research Challenges, and Emerging Frontiers — https://drive.google.com/file/d/1F3BFn64v5W64qSen5HDc-HcUQlWbTCxc/view?usp=drive_link

Deploying Large Language Models with Retrieval-Augmented Generation — https://drive.google.com/file/d/1Olv00dKmM-L4tUH4Q9hwHeYu_aZSFfiq/view?usp=drive_link

A Systematic Review of Key Retrieval-Augmented Generation (RAG) Systems: Progress, Gaps, and Future Directions — https://drive.google.com/file/d/17-Of2qPhUv1Ud2j9F68eZVX2gtmhXfnm/view?usp=drive_link

A Systematic Literature Review of Retrieval-Augmented Generation: Techniques, Metrics, and Challenges — https://drive.google.com/file/d/13vWtFkgMkiOw4ZLnNXAvcaNWib3v92Jq/view?usp=drive_link

Retrieval-Augmented Generation in Industry: An Interview Study on Use Cases, Requirements, Challenges, and Evaluation — https://drive.google.com/file/d/1oA3T-Erev0Mxziecu0LTc5O6Zy6f1ekq/view?usp=sharing

Retrieval-Augmented Code Generation: A Survey with Focus on Repository-Level Approaches — https://drive.google.com/file/d/14GK5lXgNpu0r6Cp5O6Zy6f1ekq/view?usp=sharing

Retrieval-Augmented Text Generation: Methods, Challenges, and Applications — https://drive.google.com/file/d/1RhabkjaGTjXS3Ai1sf-gkv9z_JNKub64/view?usp=sharing

Seven Failure Points When Engineering a Retrieval Augmented Generation System — https://drive.google.com/file/d/1LuR95_xZCAHtW7m8pTcVSPMYlry5IN9K/view?usp=sharing

Augmented Large Language Models for Software Engineering — https://drive.google.com/file/d/1xGQeQWGt3h2nApQcPL39sSK70Z4C80hU/view?usp=sharing

Advances in Retrieval-Augmented Generation (RAG) and Related Frameworks — https://drive.google.com/file/d/1MW57cvY_3ixV2gzUNRj6QWe15AdvNC8E/view?usp=sharing

Quais indústrias estão utilizando

Evidence Briefing

Quais indústrias estão utilizando Retrieval-Augmented Generation (RAG)?

Contexto

A arquitetura **Retrieval-Augmented Generation (RAG)** tem se consolidado como uma abordagem relevante para integrar **Large Language Models (LLMs)** com bases de conhecimento externas. Ao combinar mecanismos de recuperação de informação com geração de texto, sistemas RAG permitem que modelos de linguagem utilizem conhecimento atualizado ou específico de domínio durante a geração de respostas (Lewis et al., 2020; Gao et al., 2023).

Essa capacidade é particularmente relevante em contextos organizacionais, onde grande parte do conhecimento necessário para tomada de decisão ou execução de tarefas está distribuído em documentação interna, registros operacionais ou repositórios técnicos. Como resultado, diversas organizações têm explorado o uso de RAG para apoiar atividades relacionadas a desenvolvimento de software, operações, suporte técnico e gestão de conhecimento.

Este evidence briefing sintetiza evidências da literatura recente sobre **quais setores industriais têm adotado sistemas baseados em RAG**, com foco em aplicações relacionadas à engenharia de software e sistemas corporativos.

Evidências da literatura

A literatura indica que o uso de RAG já aparece em **diversos setores industriais**, especialmente em domínios que lidam com grandes volumes de documentação técnica ou dados operacionais. Esses contextos favorecem o uso de sistemas baseados em recuperação de informação para apoiar modelos generativos.

Entre os setores com evidência de aplicação estão **computação em nuvem, telecomunicações, cibersegurança, plataformas digitais, sistemas corporativos e serviços de tecnologia**. Em muitos casos, os sistemas RAG são utilizados para apoiar atividades como diagnóstico de falhas, consulta a documentação técnica e automação de tarefas relacionadas ao desenvolvimento de software.

Além desses domínios, estudos também identificam aplicações em setores industriais tradicionais, incluindo **automotivo e construção naval**, onde sistemas baseados em RAG são utilizados para apoiar acesso a documentação técnica e conhecimento de engenharia.

Outro grupo relevante de estudos discute aplicações de RAG de forma **transversal**, analisando arquiteturas e padrões que podem ser utilizados em diferentes setores industriais (Gao et al., 2023). Isso sugere que a tecnologia tem potencial para ser aplicada em múltiplos domínios organizacionais.

Distribuição de indústrias com evidência de uso

A Tabela 1 apresenta exemplos de setores industriais identificados na literatura que utilizam ou exploram aplicações baseadas em RAG.

Tabela 1 - Setores identificados

Indústria	Exemplos de aplicação
Computação em nuvem	análise de logs, suporte a operações, troubleshooting
Telecomunicações	suporte a pipelines CI/CD, consulta a documentação técnica
Cibersegurança	detecção de vulnerabilidades e análise de políticas de acesso
Tecnologia da Internet	revisão automática de código e completamento de código
Plataformas SaaS	busca semântica em sistemas de gestão documental
Sistemas corporativos / ERP	geração de casos de teste e automação de artefatos
Indústria automotiva	geração de scripts de teste e documentação técnica
Construção naval	acesso a conhecimento técnico e engenharia de processos
Serviços de tecnologia	suporte a desenvolvedores e engenharia de software

Interpretação das evidências

As evidências disponíveis sugerem que a adoção de RAG ocorre principalmente em setores caracterizados por **alta dependência de conhecimento técnico estruturado ou semi-estruturado**. Em ambientes de engenharia de software e operações de sistemas, por exemplo, grande parte do conhecimento necessário para resolver problemas está armazenada em documentação, logs e histórico de incidentes.

Nesse contexto, sistemas baseados em RAG permitem que modelos de linguagem consultem essas fontes de informação antes de gerar respostas, reduzindo alucinações e aumentando a confiabilidade das recomendações geradas (Lewis et al., 2020; Shuster et al., 2021).

Outro aspecto relevante é que muitas aplicações de RAG são discutidas na literatura de forma **independente de um domínio específico**, focando principalmente em arquiteturas e padrões de implementação. Isso indica que a tecnologia possui características generalizáveis que facilitam sua adoção em diferentes setores.

Conclusão

As evidências indicam que o uso de **Retrieval-Augmented Generation já se estende a múltiplos setores industriais**, particularmente aqueles que dependem fortemente de documentação técnica e gestão de conhecimento. Entre os domínios com evidência de aplicação estão computação em nuvem, telecomunicações, cibersegurança, plataformas digitais, sistemas corporativos e setores industriais tradicionais.

Essa diversidade de aplicações sugere que o RAG representa uma abordagem transversal para integrar modelos de linguagem com conhecimento organizacional, permitindo que sistemas baseados em LLMs operem de forma mais confiável em contextos industriais.

Para mais detalhes da pesquisa como fontes e artigos, consultar o White Paper - Retrieval-Augmented Generation (RAG) na Engenharia de Software desenvolvido pelo DevEficiente no link..

Referências

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W., Rocktäschel, T., Riedel, S., & Kiela, D. (2020).
Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks.
https://arxiv.org/abs/2005.11401

Gao, L., Ma, X., Lin, J., & Callan, J. (2023).
Retrieval-Augmented Generation for Large Language Models: A Survey.
https://arxiv.org/abs/2312.10997

Shuster, K., Poff, S., Chen, M., Kiela, D., & Weston, J. (2021).
Retrieval Augmentation Reduces Hallucination in Conversation.
https://arxiv.org/abs/2104.07567

Em quais etapas do Processo de Sw

Evidence Briefing

Em quais etapas do desenvolvimento de software o RAG está sendo utilizado?

Contexto

O uso de **Large Language Models (LLMs)** tem se expandido rapidamente na engenharia de software, especialmente em atividades como geração de código, análise de artefatos técnicos e suporte ao desenvolvimento. No entanto, quando utilizados isoladamente, esses modelos frequentemente carecem de conhecimento específico de domínio, como APIs internas, documentação organizacional ou histórico de incidentes.

A arquitetura **Retrieval-Augmented Generation (RAG)** surge como uma abordagem para superar essa limitação ao integrar mecanismos de recuperação de informação com modelos generativos. Ao permitir que modelos de linguagem consultem bases externas de conhecimento antes de gerar respostas, sistemas RAG conseguem fornecer suporte mais contextualizado em tarefas de engenharia de software (Lewis et al., 2020; Gao et al., 2023).

A literatura recente tem explorado o uso dessa arquitetura em diferentes atividades do ciclo de desenvolvimento de software, demonstrando que o RAG pode apoiar desde tarefas de programação até atividades relacionadas à operação e manutenção de sistemas.

Evidências da literatura

A análise da literatura mostra que o RAG tem sido aplicado em **diversas etapas do ciclo de desenvolvimento de software**, com maior concentração em atividades relacionadas ao desenvolvimento e testes.

De forma geral, as aplicações podem ser agrupadas em cinco categorias principais:

1. desenvolvimento de software
2. testes e garantia de qualidade
3. operações e suporte a incidentes
4. pipelines de integração e entrega contínua
5. segurança de software

Essas aplicações exploram a capacidade do RAG de integrar modelos de linguagem com artefatos típicos da engenharia de software, como repositórios de código, documentação técnica e registros de execução de sistemas.

Desenvolvimento de software

Uma das aplicações mais frequentes do RAG ocorre em atividades relacionadas ao **desenvolvimento de software**, incluindo geração e completamento de código, revisão automática de código e consulta a repositórios de software.

Nesses contextos, sistemas baseados em RAG permitem que modelos de linguagem recuperem informações relevantes de repositórios privados ou documentação técnica antes de gerar sugestões de código. Isso possibilita que os modelos utilizem conhecimento contextual sobre o projeto, reduzindo erros e aumentando a utilidade das recomendações geradas.

Estudos mostram que a integração entre recuperação de informação e geração de código pode melhorar significativamente o desempenho de sistemas de geração de código em ambientes organizacionais (Izacard & Grave, 2021; Gao et al., 2023).

Testes e garantia de qualidade

Outra área com evidência de uso de RAG é **testes de software**. Em alguns estudos, sistemas RAG são utilizados para gerar automaticamente casos de teste a partir de requisitos ou documentação técnica(MIZOGUCHI et al., 2025;HARIHARAN, 2026;HARIHARAN et al., 2025;ZHANG et al., 2025).

Esses sistemas recuperam informações relevantes sobre funcionalidades do sistema e utilizam essas evidências como contexto para gerar cenários de teste ou scripts de automação. Essa abordagem pode reduzir esforço manual na criação de testes e aumentar a cobertura de testes em projetos de software.

Operações e suporte a incidentes

O RAG também tem sido utilizado em atividades relacionadas à **operação e manutenção de sistemas**, particularmente no diagnóstico de falhas e análise de logs(Duan et al. 2025).

Em ambientes de produção, grande parte do conhecimento necessário para resolver problemas está distribuída em documentação técnica, histórico de incidentes e registros de execução de sistemas. Sistemas baseados em RAG permitem recuperar essas informações e utilizá-las como contexto para explicar erros ou sugerir possíveis soluções.

Essa abordagem pode auxiliar equipes de operações a reduzir o tempo necessário para identificar causas de problemas e restaurar serviços.

Integração e entrega contínua (CI/CD)

Alguns estudos relatam o uso de RAG em pipelines de **integração e entrega contínua**, especialmente para interpretar falhas em builds ou pipelines de deploy(XU et al., 2025;CHAUDHARY et al., 2024).

Nesses casos, sistemas RAG recuperam informações de logs, documentação de infraestrutura e histórico de incidentes para fornecer explicações ou recomendações relacionadas a falhas em pipelines de CI/CD.

Essa aplicação demonstra o potencial da arquitetura RAG para apoiar atividades de engenharia de software que envolvem grande volume de dados operacionais.

Segurança de software

A literatura também identifica aplicações de RAG em **segurança de software**, incluindo análise de vulnerabilidades e interpretação de requisitos de segurança(SAJU; MUHTADI; AZIM, 2026;ABOUKADRI et al., 2025).

Em alguns casos, sistemas RAG são utilizados para recuperar padrões conhecidos de vulnerabilidades ou políticas de segurança antes de gerar recomendações ou análises sobre código ou requisitos.

Essas aplicações demonstram como a integração entre recuperação de conhecimento e geração de texto pode apoiar tarefas que exigem alta precisão e contextualização.

Síntese das evidências

A Tabela 1 resume as principais etapas do ciclo de desenvolvimento de software nas quais sistemas RAG têm sido aplicados.

Etapa da engenharia de software	Exemplos de aplicação
Desenvolvimento	geração de código, completamento de código, revisão de código
Testes	geração de casos de teste, automação de testes
Operações	diagnóstico de falhas, análise de logs
CI/CD	interpretação de falhas em pipelines
Segurança	detecção de vulnerabilidades, análise de políticas

Interpretação das evidências

As evidências sugerem que o RAG tem maior impacto em atividades que envolvem **acesso intensivo a conhecimento técnico distribuído**, como documentação, logs ou histórico de incidentes.

Nesses contextos, a capacidade de recuperar informações relevantes antes da geração de respostas permite que modelos de linguagem forneçam recomendações mais contextualizadas e confiáveis.

Além disso, os resultados indicam que o RAG pode apoiar diferentes fases do ciclo de desenvolvimento de software, o que reforça seu potencial como tecnologia transversal para integração de inteligência artificial em processos de engenharia de software.

Conclusão

A literatura indica que sistemas baseados em **Retrieval-Augmented Generation estão sendo aplicados em múltiplas etapas do ciclo de desenvolvimento de software**, incluindo desenvolvimento, testes, operações, CI/CD e segurança.

Essas aplicações demonstram que a arquitetura RAG pode apoiar diferentes atividades relacionadas à construção e manutenção de sistemas de software, especialmente em contextos que exigem acesso a conhecimento organizacional.

No entanto, a efetividade dessas soluções depende da qualidade da base de conhecimento utilizada e da integração adequada entre mecanismos de recuperação e modelos de linguagem.

Para mais detalhes da pesquisa como fontes e artigos, consultar o White Paper - Retrieval-Augmented Generation (RAG) na Engenharia de Software desenvolvido pelo DevEficiente no link..

Referências

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W., Rocktäschel, T., Riedel, S., & Kiela, D. (2020).
Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks.
https://arxiv.org/abs/2005.11401

Gao, L., Ma, X., Lin, J., & Callan, J. (2023).
Retrieval-Augmented Generation for Large Language Models: A Survey.
https://arxiv.org/abs/2312.10997

Shuster, K., Poff, S., Chen, M., Kiela, D., & Weston, J. (2021).
Retrieval Augmentation Reduces Hallucination in Conversation.
https://arxiv.org/abs/2104.07567

Izacard, G., & Grave, E. (2021).
Leveraging Passage Retrieval with Generative Models for Open Domain Question Answering.
https://arxiv.org/abs/2007.01282

Quais modelos de LLM

Evidence Briefing

Quais modelos de LLM estão sendo utilizados em conjunto com RAG?

Contexto

A arquitetura **Retrieval-Augmented Generation (RAG)** combina modelos de linguagem com mecanismos de recuperação de informação, permitindo que sistemas baseados em inteligência artificial consultem bases de conhecimento externas antes de gerar respostas. Esse paradigma tem sido amplamente utilizado para melhorar a confiabilidade de **Large Language Models (LLMs)**, especialmente em cenários que exigem acesso a conhecimento específico de domínio ou dados atualizados (Lewis et al., 2020; Gao et al., 2023).

Em aplicações industriais, particularmente na engenharia de software, modelos de linguagem são frequentemente integrados a pipelines RAG para permitir o acesso a artefatos organizacionais, como documentação técnica, repositórios de código e logs operacionais. Nesse contexto, a escolha do **modelo de linguagem utilizado no componente de geração** torna-se um fator relevante para o desempenho e viabilidade do sistema.

Este evidence briefing sintetiza evidências da literatura recente sobre **quais modelos de linguagem têm sido utilizados em sistemas RAG**, com foco em aplicações relacionadas à engenharia de software.

Evidências da literatura

A análise da literatura indica que diferentes famílias de LLMs têm sido utilizadas em sistemas RAG. Esses modelos podem ser agrupados em três categorias principais: modelos proprietários de grande escala, modelos open-source e modelos emergentes utilizados em estudos comparativos.

Modelos proprietários de grande escala

Os modelos da família **GPT** aparecem com frequência em aplicações baseadas em RAG. Estudos relatam o uso de modelos como **GPT-3.5, GPT-4 e GPT-4o** como motor de geração em sistemas que integram recuperação de conhecimento corporativo ou documentação técnica. Esses modelos são frequentemente escolhidos devido à sua elevada capacidade de geração textual e à maturidade do ecossistema de ferramentas disponíveis para integração com pipelines de RAG (Gao et al., 2023).

A disponibilidade de APIs e infraestrutura associada a esses modelos facilita sua utilização em aplicações industriais, especialmente em sistemas de suporte a desenvolvedores e análise de documentação técnica.

Modelos open-source

Outra categoria importante envolve **modelos open-source**, que têm sido amplamente utilizados em arquiteturas RAG quando organizações precisam operar soluções em ambientes privados ou com maior controle sobre dados e infraestrutura.

Entre esses modelos, destacam-se os modelos da família **LLaMA**, incluindo variantes como **LLaMA-2** e **CodeLlama**. Esses modelos permitem implantação local e têm sido utilizados em sistemas que integram recuperação de conhecimento organizacional, como chatbots técnicos e ferramentas de consulta a documentação interna (Touvron et al., 2023).

Além disso, modelos open-source mais recentes, como **Mistral**, também aparecem em arquiteturas RAG voltadas a tarefas de engenharia de software e análise de artefatos técnicos.

Modelos emergentes e especializados

A literatura recente também mostra o uso de modelos emergentes em arquiteturas RAG. Um exemplo é a família **Qwen**, cujas variantes têm sido utilizadas em aplicações relacionadas à análise de logs, suporte a operações e recuperação de conhecimento técnico (Bai et al., 2023).

Outros modelos, como **Claude**, **Gemini** e **DeepSeek**, aparecem principalmente em estudos comparativos que avaliam diferentes LLMs como componentes de geração em sistemas RAG. Nesses estudos, os modelos são avaliados em termos de qualidade de resposta, custo computacional e capacidade de raciocínio em tarefas específicas.

Síntese das evidências

A Tabela 1 apresenta exemplos de estudos que utilizam diferentes modelos de linguagem em arquiteturas RAG.

Estudo	Modelo(s) de LLM	Contexto de aplicação
Lewis et al. (2020)	BART	Introdução da arquitetura RAG
Shuster et al. (2021)	BlenderBot	Conversational AI com retrieval
Gao et al. (2023)	GPT, LLaMA, T5, BART	Survey sobre arquiteturas RAG

Touvron et al. (2023)	LLaMA-2	Modelos open-source utilizados em aplicações RAG
Bai et al. (2023)	Qwen	Modelos utilizados em pipelines de recuperação e geração

Interpretação das evidências

As evidências disponíveis indicam que **não existe um único modelo dominante em sistemas RAG**. Em vez disso, diferentes famílias de modelos são utilizadas dependendo das necessidades do sistema.

Modelos proprietários, como os da família GPT, são frequentemente escolhidos quando a prioridade é maximizar a qualidade da geração e aproveitar ecossistemas maduros de ferramentas. Por outro lado, modelos open-source como LLaMA e Mistral são preferidos quando organizações precisam operar sistemas em ambientes privados ou reduzir dependência de provedores externos.

Além disso, a literatura sugere que o desempenho de sistemas RAG depende não apenas do modelo de linguagem utilizado, mas também da qualidade do mecanismo de recuperação e da estrutura da base de conhecimento (Lewis et al., 2020; Izacard & Grave, 2021).

Conclusão

A literatura mostra que arquiteturas RAG utilizam uma variedade de modelos de linguagem, incluindo tanto modelos proprietários quanto alternativas open-source. Entre as famílias mais frequentemente utilizadas estão GPT, LLaMA e Qwen, além de modelos emergentes avaliados em estudos comparativos.

Esses resultados indicam que a escolha do modelo de linguagem em sistemas RAG depende fortemente do contexto de aplicação, dos requisitos de governança de dados e das restrições operacionais do sistema.

Para mais detalhes da pesquisa como fontes e artigos, consultar o White Paper - Retrieval-Augmented Generation (RAG) na Engenharia de Software desenvolvido pelo DevEficiente no link..

Referências

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W., Rocktäschel, T., Riedel, S., & Kiela, D. (2020). Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. https://arxiv.org/abs/2005.11401

Gao, L., Ma, X., Lin, J., & Callan, J. (2023). Retrieval-Augmented Generation for Large Language Models: A Survey. https://arxiv.org/abs/2312.10997

Shuster, K., Poff, S., Chen, M., Kiela, D., & Weston, J. (2021). Retrieval Augmentation Reduces Hallucination in Conversation. https://arxiv.org/abs/2104.07567

Izacard, G., & Grave, E. (2021). Leveraging Passage Retrieval with Generative Models for Open Domain Question Answering. https://arxiv.org/abs/2007.01282

Touvron, H., et al. (2023). LLaMA 2: Open Foundation and Fine-Tuned Chat Models. https://arxiv.org/abs/2307.09288

Bai, J., et al. (2023).Qwen Technical Report. https://arxiv.org/abs/2309.16609

Quais modelos de embeddings

Evidence Briefing

Quais modelos de embeddings estão sendo utilizados com RAG?

Contexto

A arquitetura **Retrieval-Augmented Generation (RAG)** depende de um pipeline que integra recuperação de informação e geração de texto. Um dos componentes centrais desse pipeline é o uso de **embeddings**, que representam documentos e consultas como vetores em um espaço semântico. Esses vetores permitem que sistemas recuperem informações relevantes com base em similaridade semântica antes de fornecer contexto para o modelo de linguagem gerar respostas (Lewis et al., 2020; Gao et al., 2023).

A escolha do modelo de embeddings tem impacto direto na qualidade da recuperação de informação, influenciando a precisão das evidências fornecidas ao modelo generativo. Como resultado, diferentes estudos têm explorado diversas famílias de embeddings em arquiteturas RAG, variando desde modelos generalistas até embeddings especializados para domínios específicos, como código-fonte ou logs de sistemas.

Este evidence briefing sintetiza evidências da literatura sobre **quais modelos de embeddings têm sido utilizados em sistemas RAG**, com foco em aplicações relacionadas à engenharia de software e sistemas corporativos.

Evidências da literatura

A literatura indica que diferentes famílias de embeddings são utilizadas em sistemas RAG. De forma geral, essas abordagens podem ser agrupadas em quatro categorias principais:

1. embeddings generalistas baseados em transformers
2. embeddings otimizados para recuperação de informação
3. embeddings proprietários de plataformas comerciais
4. embeddings especializados para código

Embeddings baseados em transformers

Uma das famílias mais amplamente utilizadas em sistemas RAG é composta por embeddings derivados de modelos **transformer** treinados para representar texto em espaços semânticos densos.

Entre os exemplos mais utilizados estão modelos da família **Sentence Transformers**, incluindo variantes como **all-MiniLM-L6-v2** e **all-distilroberta-v1**. Esses modelos são amplamente utilizados devido à sua eficiência computacional e facilidade de integração em pipelines de busca semântica.

Esses embeddings são frequentemente utilizados em sistemas RAG que implementam recuperação baseada em similaridade vetorial em bases documentais ou bases de conhecimento organizacionais.

Embeddings otimizados para recuperação

Outra categoria importante envolve embeddings projetados especificamente para tarefas de recuperação de informação.

Entre esses modelos, destaca-se o **Dense Passage Retrieval (DPR)**, que utiliza dois encoders para representar consultas e documentos em um espaço vetorial compartilhado, permitindo recuperação eficiente de passagens relevantes (Karpukhin et al., 2020).

Além disso, abordagens como **Contriever** e **ColBERT** também aparecem na literatura de RAG como alternativas para melhorar a qualidade da recuperação semântica.

Esses modelos são particularmente relevantes em aplicações que exigem recuperação precisa de informações em grandes coleções documentais.

Embeddings proprietários

Alguns sistemas RAG utilizam **embeddings fornecidos por plataformas comerciais**, como os modelos disponibilizados pela OpenAI.

Entre esses modelos, destaca-se o **text-embedding-3-large**, que é frequentemente utilizado em pipelines RAG que utilizam modelos GPT como componente de geração.

Esses embeddings são frequentemente escolhidos devido à sua integração com APIs comerciais e ao suporte de frameworks utilizados para construir aplicações baseadas em LLMs.

Embeddings especializados para código

Em aplicações relacionadas à engenharia de software, também surgem embeddings especializados para representar **código-fonte**.

Modelos como **CodeBERT**, **UniXcoder** e **CodeT5** foram projetados para capturar relações semânticas em código e têm sido utilizados em sistemas RAG que realizam busca em repositórios de software ou suporte a tarefas de programação (Feng et al., 2020).

Esses embeddings permitem que sistemas RAG recuperem trechos de código relevantes antes de gerar sugestões ou explicações para desenvolvedores.

Síntese das evidências

A Tabela 1 resume as principais famílias de embeddings identificadas na literatura.

Categoria	Exemplos de modelos
Transformers generalistas	all-MiniLM-L6-v2, distilroberta embeddings
Recuperação densa	DPR, Contriever, ColBERT
Embeddings proprietários	text-embedding-3-large
Embeddings para código	CodeBERT, UniXcoder, CodeT5

Interpretação das evidências

As evidências indicam que a escolha do modelo de embeddings depende fortemente do **tipo de dados utilizado no sistema RAG**.

Em aplicações voltadas para documentação textual, embeddings generalistas baseados em transformers costumam ser suficientes para capturar relações semânticas relevantes. Por outro lado, em aplicações que envolvem grandes bases documentais, modelos especializados em recuperação de informação podem melhorar significativamente a precisão da busca.

Além disso, em contextos específicos como engenharia de software, embeddings treinados para representar código-fonte podem oferecer vantagens importantes ao permitir recuperação mais precisa de trechos de código relevantes.

Esses resultados reforçam a importância da etapa de recuperação dentro da arquitetura RAG, indicando que a qualidade do embedding pode ter impacto significativo no desempenho final do sistema.

Conclusão

A literatura mostra que sistemas RAG utilizam uma variedade de modelos de embeddings, incluindo modelos generalistas baseados em transformers, embeddings otimizados para recuperação de informação, embeddings proprietários e embeddings especializados para código.

Essas evidências indicam que a escolha do modelo de embeddings em arquiteturas RAG depende principalmente do domínio da aplicação e do tipo de dados utilizados no sistema.

Como resultado, a seleção adequada de embeddings representa um fator crítico para o sucesso de aplicações baseadas em RAG, especialmente em contextos organizacionais que dependem de recuperação eficiente de conhecimento.

Para mais detalhes da pesquisa como fontes e artigos, consultar o White Paper - Retrieval-Augmented Generation (RAG) na Engenharia de Software desenvolvido pelo DevEficiente no link..

Referências

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W., Rocktäschel, T., Riedel, S., & Kiela, D. (2020).
Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks.
https://arxiv.org/abs/2005.11401

Gao, L., Ma, X., Lin, J., & Callan, J. (2023).
Retrieval-Augmented Generation for Large Language Models: A Survey.
https://arxiv.org/abs/2312.10997

Karpukhin, V., Oguz, B., Min, S., Lewis, P., Wu, L., Edunov, S., Chen, D., & Yih, W. (2020).
Dense Passage Retrieval for Open-Domain Question Answering.
https://arxiv.org/abs/2004.04906

Feng, Z., Guo, D., Tang, D., Duan, N., Feng, X., Gong, M., Shou, L., Qin, B., Liu, T., & Jiang, D. (2020).
CodeBERT: A Pre-Trained Model for Programming and Natural Languages.
https://arxiv.org/abs/2002.08155

Quais arquitetura?

Evidence Briefing

Quais arquiteturas de implementação de RAG são adotadas?

Contexto

A arquitetura **Retrieval-Augmented Generation (RAG)** combina modelos de linguagem com mecanismos de recuperação de informação para permitir que sistemas baseados em inteligência artificial utilizem bases de conhecimento externas durante a geração de respostas. Esse paradigma foi introduzido por Lewis et al. (2020) e tem se consolidado como uma das principais abordagens para integrar **Large Language Models (LLMs)** com dados específicos de domínio.

Em sistemas RAG, o pipeline geralmente envolve etapas de **indexação de documentos, recuperação de informações relevantes e geração condicionada ao contexto recuperado**. No entanto, à medida que aplicações industriais se tornam mais complexas, diferentes arquiteturas têm sido desenvolvidas para melhorar a precisão da recuperação, reduzir ruído nas evidências recuperadas e lidar com múltiplas fontes de dados (Gao et al., 2023).

Este evidence briefing sintetiza evidências da literatura sobre **as principais arquiteturas utilizadas na implementação de sistemas RAG**, com foco em aplicações relacionadas à engenharia de software e sistemas corporativos.

Evidências da literatura

A literatura identifica diversas variações da arquitetura RAG, que podem ser agrupadas em seis padrões principais de implementação:

1. pipeline RAG padrão
2. recuperação em dois estágios (retrieve + rerank)
3. recuperação híbrida
4. RAG guiado por intenção
5. RAG multi-fonte
6. arquiteturas baseadas em grafos ou agentes

Pipeline RAG padrão

A arquitetura mais básica de RAG consiste em um pipeline simples no qual documentos são indexados em um banco vetorial e recuperados com base na similaridade semântica de

uma consulta. Os documentos recuperados são então utilizados como contexto para a geração de respostas por um modelo de linguagem (Lewis et al., 2020).

Esse pipeline pode ser descrito de forma simplificada como:

consulta → recuperação → geração

Embora seja relativamente simples, esse modelo já permite que sistemas baseados em LLMs utilizem conhecimento externo durante a geração de respostas.

Recuperação em dois estágios

Em sistemas RAG mais avançados, a recuperação de documentos é frequentemente realizada em duas etapas. Primeiro, um mecanismo de busca recupera um conjunto amplo de documentos candidatos. Em seguida, um modelo adicional de **reranking** é utilizado para selecionar os documentos mais relevantes antes da geração da resposta.

Esse padrão permite melhorar a precisão da recuperação e reduzir o impacto de documentos irrelevantes no processo de geração (Gao et al., 2023).

Recuperação híbrida

Outra arquitetura comum em sistemas RAG envolve a combinação de **busca lexical e busca semântica**. Nesse modelo, métodos tradicionais de recuperação de informação, como BM25 ou TF-IDF, são combinados com busca baseada em embeddings.

A fusão dessas abordagens permite equilibrar duas propriedades importantes da recuperação de informação: a capacidade de capturar similaridade semântica e a capacidade de recuperar termos específicos ou identificadores técnicos.

Esse tipo de arquitetura tem sido utilizado em sistemas que lidam com documentação técnica ou código-fonte.

RAG guiado por intenção

Em algumas aplicações industriais, sistemas RAG incorporam mecanismos adicionais para identificar a intenção da consulta do usuário antes de realizar a recuperação de documentos(HUANG et al., 2025;XU et al., 2025;ABOUKADRI et al., 2025).

Nessas arquiteturas, o sistema pode realizar etapas como:

- classificação da intenção da consulta
- extração de metadados relevantes
- reescrita da consulta

Esses componentes permitem melhorar a qualidade da recuperação de informações e lidar com consultas ambíguas ou incompletas.

RAG multi-fonte

Outro padrão identificado na literatura envolve sistemas que realizam recuperação de informações em **múltiplas fontes de dados**. Em ambientes organizacionais, o conhecimento relevante pode estar distribuído em diferentes tipos de repositórios, como documentação técnica, bases de conhecimento, logs operacionais e repositórios de código.(XU et al., 2025)

Arquiteturas RAG multi-fonte permitem recuperar evidências dessas diferentes fontes e combiná-las como contexto para a geração de respostas.

Esse tipo de arquitetura é particularmente relevante em aplicações relacionadas à engenharia de software e operações de sistemas.

Arquiteturas baseadas em grafos ou agentes

Estudos mais recentes exploram arquiteturas RAG mais complexas que utilizam **estruturas de grafos ou sistemas multiagente** para organizar e recuperar conhecimento(YANG; CHEN, 2025;HARIHARAN et al., 2025;HARIHARAN, 2026).

Em sistemas baseados em grafos, relações entre diferentes artefatos de software, como arquivos, funções e dependências, são representadas explicitamente, permitindo recuperar contexto estrutural para tarefas como revisão de código ou análise de impacto.

Além disso, algumas arquiteturas utilizam múltiplos agentes especializados para executar diferentes etapas do pipeline, como recuperação de informação, filtragem de documentos e geração de respostas.

Síntese das evidências

A Tabela 1 resume as principais arquiteturas de implementação identificadas na literatura.

Arquitetura	Características
Pipeline RAG padrão	recuperação simples seguida de geração
Recuperação em dois estágios	recuperação inicial + reranking
Recuperação híbrida	combinação de busca lexical e semântica
RAG guiado por intenção	detecção e reescrita de consultas
RAG multi-fonte	recuperação em múltiplos repositórios
RAG baseado em grafos ou agentes	uso de grafos ou múltiplos agentes para organizar conhecimento

Interpretação das evidências

As evidências indicam que arquiteturas RAG têm evoluído de pipelines relativamente simples para sistemas mais complexos capazes de lidar com diferentes tipos de dados e consultas.

Em aplicações industriais, a principal motivação para essa evolução é a necessidade de recuperar conhecimento relevante em ambientes onde a informação está distribuída em múltiplos repositórios e formatos.

Além disso, estudos mostram que melhorias na etapa de recuperação como reranking ou recuperação híbrida podem ter impacto significativo na qualidade das respostas geradas por modelos de linguagem (Lewis et al., 2020; Gao et al., 2023).

Conclusão

A literatura mostra que sistemas RAG podem ser implementados utilizando diferentes arquiteturas, variando desde pipelines simples de recuperação e geração até sistemas mais complexos que incorporam múltiplas fontes de dados, reranking e estruturas de grafos.

Essas evidências indicam que a escolha da arquitetura RAG depende principalmente do tipo de aplicação, da complexidade das fontes de dados e dos requisitos de precisão na recuperação de informações.

Para mais detalhes da pesquisa como fontes e artigos, consultar o White Paper - Retrieval-Augmented Generation (RAG) na Engenharia de Software desenvolvido pelo DevEficiente no link..

Referências

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W., Rocktäschel, T., Riedel, S., & Kiela, D. (2020).
Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks.
https://arxiv.org/abs/2005.11401

Gao, L., Ma, X., Lin, J., & Callan, J. (2023).
Retrieval-Augmented Generation for Large Language Models: A Survey.
https://arxiv.org/abs/2312.10997