Análise de Eficiência Computacional em Sistemas RAG com Busca Vetorial: Um Estudo de Caso no *ProbY*

Luis Oliveira[1], Douglas Zanini[1], Alex Camargo[1]

[1]*ProbY*: *A Smart Platform for Global Problems*
Bagé, RS – Brasil

{luis, douglas, alex}@proby.online

Resumo. *Este trabalho apresenta uma análise da integração entre Retrieval-Augmented Generation (RAG) e busca vetorial sob a perspectiva de eficiência computacional. O estudo avalia o desempenho das etapas de geração de embeddings, recuperação vetorial e geração de respostas por modelos de linguagem. A partir de um benchmarking simplificado, são avaliadas métricas de latência e custo do pipeline. Os resultados indicam que a recuperação vetorial apresenta baixa latência, enquanto a geração de texto concentra a maior parte do tempo de processamento.*

1. Introdução

Nos últimos anos, os modelos de linguagem amplos (*Large Language Models* - LLMs) tornaram-se amplamente utilizados em aplicações de processamento de linguagem natural, incluindo sistemas de perguntas e respostas, geração de conteúdo e assistentes inteligentes (Gupta et al., 2024). Apesar de sua capacidade de gerar textos coerentes, esses modelos possuem limitações relacionadas à atualização de conhecimento e à precisão concreta das respostas geradas. Ji et al. (2023) explicam que como o conhecimento é incorporado durante o treinamento, os modelos podem apresentar dificuldades ao lidar com informações recentes ou específicas de determinados domínios, além de ocasionalmente produzir respostas plausíveis, porém incorretas. Uma abordagem amplamente utilizada para mitigar essas limitações é o uso de arquiteturas *Retrieval-Augmented Generation* (RAG). Nesse modelo híbrido, um componente de recuperação de informação busca documentos relevantes em uma base externa de conhecimento, enquanto um modelo generativo utiliza esses documentos para produzir respostas mais contextualizadas e fundamentadas (Zhao et al., 2026). Tal combinação permite unir a capacidade de recuperação eficiente de informação com a habilidade de geração textual dos modelos de linguagem. A partir disso, a análise de eficiência computacional se torna um aspecto relevante, especialmente em aplicações que demandam respostas em tempo real. Esses aspectos aproximam o problema da área de Computação de Alto Desempenho, sobretudo no que se refere à otimização de latência, escalabilidade e uso eficiente de recursos computacionais.

Este trabalho apresenta uma análise sobre RAG integrada com busca vetorial, com o objetivo de explorar aspectos de processamento de alto desempenho e eficiência computacional por meio de um *benchmarking* simplificado. Nesse sentido, a pesquisa contextualiza uma aplicação prática na plataforma *ProbY: A Smart Platform for Global Problems*[1], utilizada como estudo de caso. Embora os autores façam parte da empresa responsável pela plataforma, a análise apresentada busca extrapolar esse contexto específico, discutindo implicações aplicáveis a diferentes domínios.

1 Disponível em: https://proby.online

2. *Pipeline* Experimental

O *pipeline* experimental[2] é composto por quatro etapas principais: (1) preparação dos documentos, (2) geração de *embeddings*, (3) indexação vetorial e (4) recuperação de informações.

A vetorização dos documentos foi realizada utilizando o modelo *text-embedding-ada-002*, aplicado após as etapas de extração de texto e segmentação em *chunks*, com posterior inserção dos vetores no banco *Qdrant*. A geração de respostas foi realizada com um modelo da família GPT que tem como foco a qualidade das repsostas, denominado *gpt-4o*. A indexação vetorial foi realizada com o banco *Qdrant*, utilizando o algoritmo de busca aproximada HNSW (*Hierarchical Navigable Small World*), com métrica de similaridade baseada em cosseno. As consultas utilizam estratégia *top-k* com pontuação por similaridade, podendo incluir filtragem por metadados (por exemplo, *company_id*) para isolamento lógico dos dados em cenários *multi-tenant*. Os experimentos foram executados em uma infraestrutura distribuída composta por três componentes principais. A aplicação responsável pela orquestração do pipeline RAG (ingestão, recuperação e geração) foi hospedada em um servidor na plataforma *IONOS*, executando o *Ubuntu 24.04.4 LTS*, com 4 GB de RAM e 2 *vCores AMD EPYC Milan*. O banco vetorial foi provisionado no serviço gerenciado *Qdrant Cloud*, executado sobre uma infraestrutura da AWS, na região *sa-east-1* (São Paulo), utilizando um *cluster* com 1 nó, 0.5 vCPU, 1 GB de RAM e 4 GB de armazenamento. Os serviços de inferência, incluindo geração de *embeddings* e respostas, foram realizados via API da *OpenAI* com *backend* baseado em *Laravel* (PHP 8.3). Dessa forma, as medições de latência refletem o custo computacional das operações internas e os efeitos de rede entre componentes distribuídos. A Figura 1 ilustra a arquitetura conceitual de sistemas RAG e sua adaptação para o ambiente da plataforma *ProbY*, evidenciando as etapas de geração de *embeddings*, indexação vetorial e recuperação de documentos.

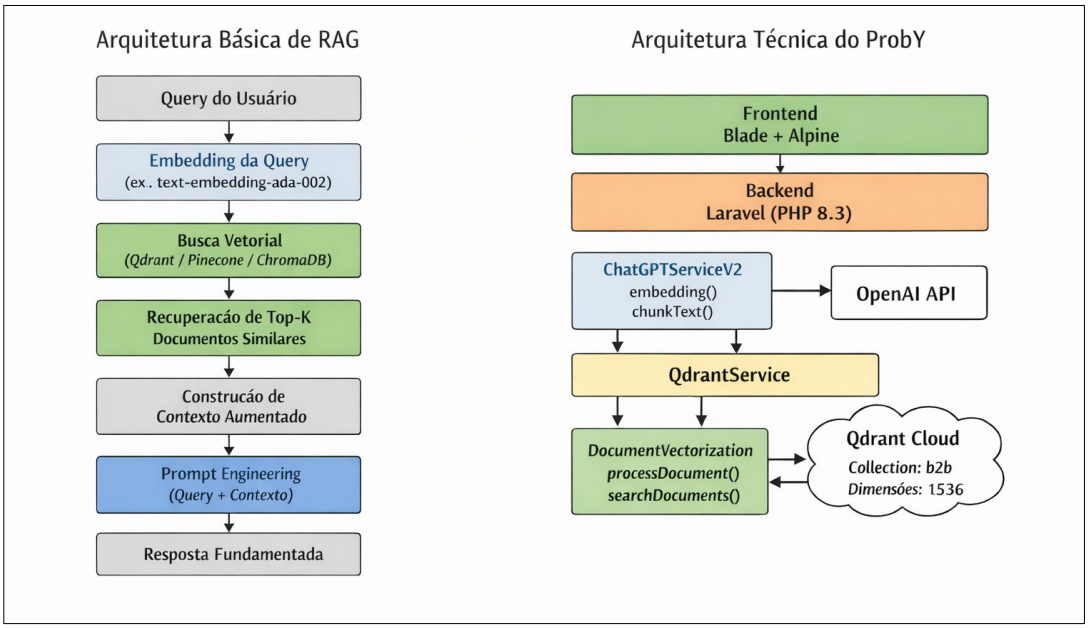

Figura 1. Arquitetura conceitual de um sistema RAG e sua implementação técnica no *pipeline* da plataforma ProbY

2 Repositório dos experimentos: https://github.com/Luis-edubr/rag-erad-proby

No *pipeline* implementado, as consultas dos usuários são inicialmente convertidas em representações vetoriais por meio do modelo de *embedding*. Esses vetores são utilizados para realizar buscas por similaridade em um banco vetorial baseado em *Qdrant*, permitindo recuperar os documentos semanticamente mais próximos. Os documentos recuperados são então utilizados como contexto para a construção do *prompt* enviado ao modelo de linguagem, permitindo gerar respostas fundamentadas nas informações disponíveis na base de conhecimento.

3. Resultados Obtidos

A Tabela 1 resume as principais métricas para cada componente da arquitetura. As medições consideram tanto cenários sem utilização de *cache* quanto cenários com mecanismos de *cache* habilitados, permitindo avaliar possíveis ganhos de desempenho em aplicações que realizam consultas repetidas ou similares. Essa análise permite interpretar o comportamento do *pipeline* sob a ótica de desempenho computacional, evidenciando características relevantes para cenários de alto desempenho.

Table 1. Métricas de performance e custo por componente da arquitetura RAG

Componente	Volume	Custo	Tempo (sem *cache*)	Tempo (com *cache*)
Embedding latency	20 *tokens* por *query*	US$ 0,10 1M *tokens*	145 ms	35 ms
Vector search latency	Busca em coleção de 5M de vetores	US$ 0,10 1M vetores	62 ms	48 ms
LLM generation latency	500 *tokens* gerados por resposta	US$ 10,00 1M *tokens*	1900 ms	1700 ms

A partir dos dados apresentados, observamos que a etapa de geração de *embeddings* apresenta redução de aproximadamente 75% na latência com uso de *cache* (145 ms para 35 ms). Na busca vetorial, a redução observada é de cerca de 23% (62 ms para 48 ms). Já na geração de respostas, a redução é mais discreta, em torno de 10% (1900 ms para 1700 ms). Considerando o tempo total do *pipeline*, a etapa de geração de texto representa mais de 85% da latência total da requisição, enquanto as etapas de recuperação e processamento vetorial possuem impacto significativamente menor.

Para contextualizar os resultados obtidos, observamos em Johnson et al. (2025) que a geração de *embeddings* em APIs comerciais geralmente apresenta latência na ordem de dezenas a poucas centenas de milissegundos por requisição, enquanto operações de busca vetorial em bases com milhões de vetores tendem a apresentar tempos inferiores a 100 ms em cenários com indexação otimizada. Por outro lado, a geração de respostas por modelos de linguagem frequentemente apresenta latência na ordem de segundos, devido à complexidade do processo generativo. Nesse sentido, os valores observados neste estudo estão alinhados com padrões reportados para sistemas RAG em ambientes reais, indicando que o desempenho obtido é compatível com o esperado para aplicações baseadas em recuperação semântica e geração de linguagem natural. Em termos de custo computacional, observamos que a geração de *embeddings* apresenta valor aproximado de US$ 0,10 por milhão de *tokens* processados, enquanto a geração de texto pelo modelo de linguagem apresenta custo médio de cerca de US$ 10,00 por milhão de *tokens* gerados. Esses valores reforçam a importância de estratégias de otimização no *pipeline*, como reutilização de *embeddings*, redução do número de chamadas ao modelo de linguagem e uso eficiente de mecanismos de recuperação.

4. Conclusão

A partir de um *pipeline* experimental simplificado, foi possível observar o comportamento de desempenho dos principais componentes envolvidos na arquitetura, incluindo a geração de *embeddings*, a recuperação de documentos por similaridade vetorial e a geração de respostas por modelos de linguagem. Os resultados obtidos indicam que a etapa de recuperação vetorial apresenta latência relativamente baixa mesmo em cenários com grandes volumes de dados, demonstrando a eficiência de estruturas de indexação vetorial e algoritmos de busca aproximada para consultas semânticas. Em contrapartida, a etapa de geração de respostas pelo modelo de linguagem representa a maior parcela do tempo total de processamento, evidenciando que a geração de texto ainda constitui o principal fator de custo computacional em sistemas baseados em RAG. A análise também demonstrou que o uso de mecanismos de *cache* pode contribuir para reduzir significativamente a latência em diferentes etapas do *pipeline*, especialmente na geração de *embeddings* e na recuperação de documentos. Esse tipo de otimização pode ser relevante em aplicações que realizam consultas recorrentes ou que operam em ambientes com alta demanda de processamento. Como principal contribuição, este trabalho evidencia, em um cenário real de aplicação, que a etapa de geração de respostas por LLMs concentra a maior parcela do custo e da latência total, enquanto a busca vetorial apresenta desempenho estável mesmo em bases de grande escala. Além disso, demonstra que otimizações simples, como uso de *cache* e redução de chamadas ao modelo generativo, podem gerar ganhos significativos de desempenho.

Como trabalhos futuros, pretendemos investigar estratégias adicionais de otimização de desempenho, além de explorar arquiteturas avançadas para recuperação e geração de contexto, incluindo abordagens híbridas de recuperação de informação, conforme discutido por Han et al. (2024). Adicionalmente, avaliaremos o impacto de diferentes configurações de modelos de linguagem na latência e no custo total do *pipeline*, considerando cenários distintos de requisitos de qualidade e desempenho.

Referências

Gupta, S.; Verma, R.; Sharma, A. (2024). A Comprehensive Survey of Retrieval-Augmented Generation (RAG). arXiv. Disponível em: https://arxiv.org/abs/2410.12837. Acesso em: 3 mar. 2026.

Ji, Z.; Lee, N.; Frieske, R.; Yu, T.; Su, D.; Xu, Y.; Ishii, E.; Bang, Y.; Madotto, A.; Fung, P. (2023). *Survey of Hallucination in Natural Language Generation*. ACM Computing Surveys. Disponível em: https://arxiv.org/pdf/2202.03629.pdf. Acesso em: 6 mar. 2026.

Johnson, J.; Douze, M.; Jégou, H. (2025). *Faiss: A Library for Efficient Similarity Search and Clustering of Dense Vectors.* arXiv. Disponível em: https://arxiv.org/pdf/2401.08281.pdf. Acesso em: 7 mar. 2026.

ProbY. (2026). ProbY – Plataforma Inteligente de Problemas Globais. Disponível em: https://proby.online. Acesso em: 6 mar. 2026.

Zhao, P.; Liu, Y.; Wang, H. (2026). Retrieval-Augmented Generation for AI-Generated Content. Springer. Disponível em: https://link.springer.com/article/10.1007/s41019-025-00335-5. Acesso em: 5 mar. 2026.