Fastcamp de Agentes Inteligentes

Relatório 06 - Prática: Embedding (II)

Lucas Augusto Nunes de Barros

Descrição da atividade

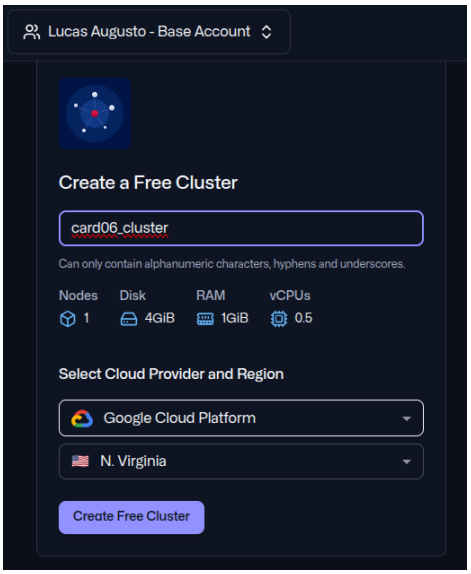
O card traz o tema de embeddings aplicado ao Qdrant, um motor de busca vetorial de alto desempenho, projetado especificamente para armazenar, gerenciar e recuperar embeddings. Diferente de bancos de dados relacionais tradicionais, que operam baseados em tabelas e correspondência exatas, o Qdrant utiliza algoritmos de indexação, como o HNSW (*Hierarchical Navigable Small World*), para realizar buscas por "vizinhos mais próximos". Essa abordagem permite ao sistema compreender a similaridade semântica entre diferentes fragmentos de texto, transformando dados não estruturados em representações numéricas que capturam o contexto e o significado da informação.

Atualmente, com o aumento no desenvolvimento de agentes, o uso de técnicas como RAG (*Retrieval-Augmented Generation*) vem se tornando cada vez mais comum, e com isso os bancos de dados vetoriais entram em cena cada vez mais. Dentro desse fluxo de trabalho o Qdrant atua como a memória de longo prazo e motor de recuperação de contexto. Quando o usuário realiza uma consulta, o sistema não depende da memória pré-treinada do modelo generativo, na verdade o LLM consulta o Qdrant para recuperar os fragmentos de informação (*chunks*) mais próximos. Esses fragmentos são então injetados no *prompt* do LLM, permitindo que o modelo dê respostas mais precisas, diminuindo a chance de alucinação.

1. Criando uma Instância Qdrant na Nuvem

Ao criar uma conta no Qdrant é possível criar uma instância gratuita na nuvem da plataforma, também é possível criar uma instância local usando docker, por exemplo. Como a plataforma disponibiliza uma cota de uso gratuito da nuvem, essa foi a opção escolhida para prosseguir com a implementação.

Criando um cluster no Qdrant



Criado o cluster em nuvem, são disponibilizados o endpoint e chave da API. Finalizando a criação do primeiro cluster, é hora de prosseguir para o curso Qdrant Essentials, disponibilizado no site oficial.

2. O Curso Qdrant Essentials

Esse curso foi desenvolvido para apresentar e consolidar o conhecimento sobre busca vetorial, iniciando da teoria básica sobre embeddings e espaço multidimensional até a implementação em produção, passando pela apresentação da plataforma e seus recursos. O curso está dividido em dias, iniciando do dia zero e indo até o dia sete. Para esse relatório foram estudados os dias zero, um e dois.

2.1 Dia Zero: Configuração e Primeiros Passos

Logo de início são apresentadas as telas e formas de interagir com as coleções dentro da plataforma, bem como a forma de interpretar cada uma das informações contidas no dashboard. Prosseguindo, são passadas instruções de como conectar ao Qdrant com uma aplicação python.

Boas práticas recomendadas pela documentação:

- Mantenha os segredos fora do código; use variáveis de ambiente ou um gerenciador de chaves secretas.
- Restrinja o acesso com listas de permissões usando IP ou redes privadas.
- Alterne as chaves de API regularmente.
- Use apenas HTTPS;

Problemas comuns:

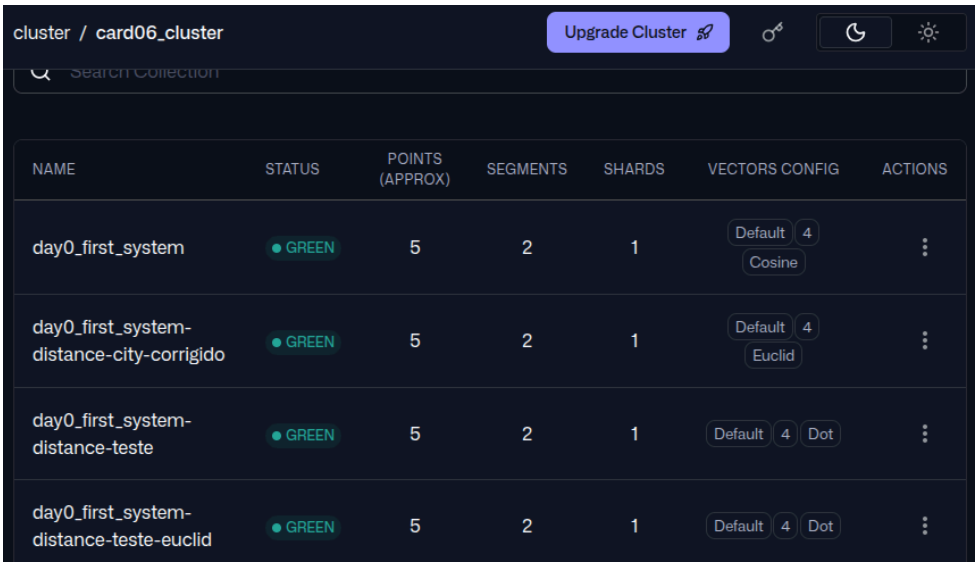
- Erro de autenticação: verifique a chave da API e o cabeçalho da chave da API.
- Erro de conexão: confirme o status do cluster e a URL da região; alguns proxies corporativos bloqueiam o TLS de saída.

No final desta etapa é apresentado um vídeo sobre a nova funcionalidade, conhecida como inferência. Essa funcionalidade basicamente permite ao usuário entrar diretamente com dados brutos e a própria plataforma Qdrant faz o trabalho de converter os dados em embeddings, sendo que antes era necessário o usuário fazer a conversão dos dados para embeddings antes de realizar o upload para a plataforma.

A seguir é apresentado o passo a passo de como implementar uma busca vetorial básica utilizando Python e Google Colab. Além de criar uma nova coleção de dados, são apresentados alguns conceitos como métricas de distância. As métricas de distância são utilizadas para medir o nível de semelhança entre os embeddings. A escolha da métrica depende da forma como os vetores são obtidos.

O dia zero finaliza com um projeto que consiste em implementar vetores de forma manual e realizar buscas dentro das coleções de vetores utilizando recursos da biblioteca Qdrant.

Coleções Criadas Manualmente no Cluster



2.2 Dia Um: Fundamentos da Busca Vetorial

Nessa etapa, somos apresentados ao núcleo do modelo de dados do Qdrant, composto por pontos, vetores e metadados (payload).

2.2.1 Pontos Vetores e Metadados

Um ponto é o registro composto por três partes

- ID
- Vetor (denso, esparso ou mutivetor)
- Payload opcional

Os vetor podem ainda ser de três tipos:

Vetores densos são os mais utilizados por aplicações com IA e ML, sendo também os tipos de vetor mais comum gerados pelo modelos de rede neural. Assim como no conceito matemático, o vetor de embeddings é uma representação dentro de uma espaço multidimensional, o que torna possível encontrar o sentido de dados não estruturados, permitindo busca semântica e outras aplicações.

Existem também os vetores esparsos, que são similares aos densos porém contendo diversos zeros, o que permite a representação na forma de uma lista de pares índice-valor. E ainda existem multi vetores, um conjunto de vetores que representa os embeddings de uma única entrada, o número de vetores por representação varia enquanto o tamanho de cada vetor é fixo.

A documentação ainda apresenta algumas comparações entre modelos geradores de embeddings e traz também algumas explicações sobre dimensionalidade dos vetores, e como isso implica no resultado final da aplicação. Um trecho interessante é sobre os metadados, ou payloads, que armazenam dados sobre os embeddings para permitir busca e filtragem. Essa combinação entre vetor e metadados permite unir a relevância semântica dos embeddings com

a lógica de negócios. Esses metadados podem armazenar texto (descrições, tags, classes), valores numéricos (datas, preços, avaliações) ou estruturas complexas (objetos aninhados, arrays).

Diferentes tipos de metadados

Tipo (Type)	Descrição	Exemplo
Keyword	Para correspondência exata de strings (ex: tags, categorias, IDs).	`category: "electronics"`
Integer	Inteiros com sinal de 64 bits para filtragem numérica.	`stock_count: 120`
Float	Números de ponto flutuante de 64 bits para preços, avaliações, etc.	`price: 19.99`
Bool	Valores verdadeiro/falso.	`in_stock: true`
Geo	Pares de latitude/longitude para consultas baseadas em localização.	`location: { "lon": 13.4050, "lat": 52.5200 }`
Datetime	Carimbos de data/hora no formato RFC 3339 para filtragem temporal.	`created_at: "2024-03-10T12:00:00Z"`
UUID	Um tipo eficiente em memória para armazenar e corresponder UUIDs.	`user_id: "550e8400-e29b-41d4-a716-446655440000"`

O Qdrant ainda permite usar condições lógicas, que podem ser aninhadas, para criar consultas complexas. Algumas delas são apresentadas abaixo:

- **must**: todas as condições devem ser satisfeitas (lógica AND)
- **should**: pelo menos uma condição deve ser satisfeita (lógica OR)
- **must_not**: nenhuma das condições deve ser satisfeita (lógica NOT)

Além das condições lógicas básicas, o Qdrant oferece mais um conjunto de condições para filtrar diferentes tipos de dados, essas condições permitem criar consultas mais precisas.

Tipos de Filtro Disponíveis no Qdrant

Tipo de Filtro	Descrição (Tradução)	Exemplo de Consulta (Query)
Match	Valor exato	`"match": {"value": "electronics"}`
Match Any	Condição OU (OR)	`"match": {"any": ["red", "blue"]}`
Match Except	Condição NÃO ESTÁ EM (NOT IN)	`"match": {"except": ["banned"]}`
Range	Intervalos numéricos	`"range": {"gte": 50, "lte": 200}`
Datetime Range	Filtragem baseada em tempo	`"range": {"gt": "2023-01-01T00:00:00Z"}`
Full Text	Correspondência de substring	`"match": {"text": "amazing service"}`
Geospatial	Baseado em localização	`"geo_radius": {"center": {...}, "radius": 10000}`
Nested	Filtragem de objetos em array	`"nested": {"key": "reviews", "filter": {...}}`
Has ID	IDs específicos	`"has_id": [1, 5, 10]`
Is Empty	Campos ausentes	`"is_empty": {"key": "discount"}`
Is Null	Valores nulos	`"is_null": {"key": "field"}`
Values Count	Comprimento do array	`"values_count": {"gt": 2}`

2.2.2 Métricas de Distância

Nesse outro tópico são apresentadas algumas métricas de distância que o Qdrant utiliza para realizar suas buscas vetoriais.

Similaridade de Cosseno

- Casos de uso comuns: incorporações NLP, pesquisa semântica, recuperação de documentos.
- Foco: direção (orientação). A magnitude é ignorada.
- A similaridade coseno mede a similaridade angular entre dois vetores. Ela se concentra em se os vetores apontam na mesma direção, em vez de se concentrar no seu comprimento. Isso se alinha bem com muitas incorporações de texto, onde o ângulo codifica o significado e o comprimento é menos importante.

Similaridade do produto escalar

- Casos de uso comuns: sistemas de recomendação, fatoração de matrizes, classificação.
- Foco: magnitude e direção.
- A similaridade do produto escalar é calculada multiplicando os valores respectivos nos dois vetores e somando esses produtos. Ao contrário do coseno, essa métrica considera o comprimento do vetor.

Distância euclidiana (distância L2)

- Casos de uso comuns: dados espaciais, detecção de anomalias, agrupamento.
- Foco: distância absoluta entre pontos.
- A distância euclidiana calcula a distância em linha reta entre dois pontos no espaço multidimensional. É frequentemente usada quando a diferença numérica exata entre vetores é importante, como em algoritmos de agrupamento (por exemplo, K-Mean).

Distância de Manhattan

- Casos de uso comuns: dados esparsos, tratamento robusto de outliers.
- Foco: distância baseada em grade (soma das diferenças absolutas).
- A distância de Manhattan é semelhante à distância euclidiana, mas calcula a distância como se estivesse se movendo ao longo das linhas da grade (horizontais e verticais).

A documentação traz também um resumo sobre quando usar cada uma das métricas.

Métrica	Aplicação Comum	Por quê?
Similaridade de Cosseno	PNL, Busca Semântica	Ignora a magnitude; foca no significado semântico (direção).
Produto Escalar	Recomendações, Classificação (Ranking)	Captura tanto a direção quanto a magnitude (importância/popularidade).
Distância Euclidiana	Dados Espaciais, Detecção de Anomalias	Mede a distância física ou numérica absoluta.
Distância de Manhattan	Dados Esparsos/Tabulares	Mais robusta a valores discrepantes (outliers) do que a Euclidiana.

2.2.3 Segmentação de Texto (Chunking Text)

Após as métricas de distância o curso traz um tópico sobre fragmentação de texto (chunking text), e explica porque a aplicação desse conceito no contexto de busca vetorial apresenta resultados positivos.

Os gráficos *Hierarchical Navigable Small World* (HNSW) estão entre os índices de melhor desempenho para busca por similaridade vetorial. Ele opera como um grafo multicamadas. O algoritmo organiza os vetores em uma hierarquia de grafos interconectados: as camadas superiores são esparsas, contendo apenas alguns pontos, enquanto as camadas inferiores tornam-se progressivamente mais densas, até a camada base (Layer 0), que contém a totalidade dos vetores armazenados.

Sem chunking:

- O contexto é muito longo e acaba sendo truncado pelo modelo, podendo perder completamente dados .
- Mesmo não truncando o vetor resultante é uma média de todos os tópicos, o que torna improvável que uma busca retorne o melhor vetor possível para consultas mais específicas.

Com chunking adequado:

- O contexto é dividido em fragmentos, cada um focado em um tópico.
- O fragmento sobre determinado parâmetro obtém o seu próprio vetor.
- O Qdrant consegue recuperar mais facilmente este fragmento específico.

Em vez de tratar os documentos como blocos monolíticos, é mais fácil trabalhar com eles quando são divididos em parágrafos, títulos, seções, páginas, e assim por diante. Seguindo esse raciocínio, os documentos são divididos e cada fragmento recebe o seu próprio vetor de embeddings, vinculado a uma ideia ou tópico específico, os metadados adicionados a cada fragmento são mais específicos e realistas.

Essa separação permite:

- Recuperação filtrada - “Mostrar apenas resultados desta seção”
- Fragmentos sensíveis ao contexto - Respostas precisas a consultas específicas
- Processamento eficiente - Sem desperdício de tokens em conteúdo irrelevante

Comparação entre as estratégias de segmentação de texto:

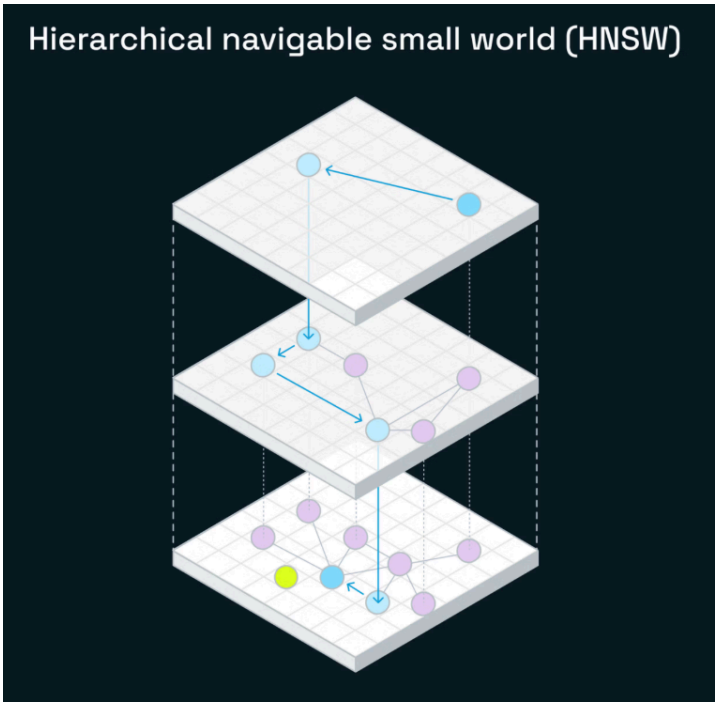
Método	Ponto Forte	Trade-off (Desvantagem)	Melhor Para
Tamanho Fixo (Fixed-Size)	Simples, fragmentos previsíveis.	Ignora a estrutura, quebra o significado.	Texto bruto ou não estruturado.
Sentença (Sentence)	Preserva pensamentos completos.	Tamanhos inconsistentes.	Sistemas RAG, Perguntas e Respostas (Q&A).
Parágrafo (Paragraph)	Alinha-se com unidades semânticas.	Grande variação no comprimento.	Documentação, manuais, conteúdo instrucional.
Janela Deslizante (Sliding Window)	Mantém o contexto completo.	Redundante, computacionalmente pesado.	Reclassificação (Reranking), recuperação de alta revocação.
Recursivo (Recursive)	Flexível, lida com entradas desorganizadas.	Heurístico, às vezes instável.	Conteúdo extraído da web (scraped), fontes mistas.
Semântico (Semantic)	Alta qualidade, ciente do significado.	Mais lento, intensivo em recursos.	Jurídico, pesquisa, QA crítico.

2.3 Dia Dois: Fundamentos da Indexação HNSW

Para compreender o funcionamento do HNSW, basta pensar que ele é similar ao sistema de organização de uma grande biblioteca, onde para buscar um livro específico possuindo apenas uma breve descrição do seu conteúdo, é preciso começar filtrando de forma mais ampla e ir se tornando mais específico até encontrar o livro desejado.

Caso o acervo da biblioteca estivesse organizado aleatoriamente, seria necessário verificar cada livro individualmente, o método de força bruta, extremamente lento e caro em recursos. Para mitigar esse tipo de situação, as bibliotecas utilizam uma estrutura lógica projetada para guiar o leitor: inicia-se pela escolha de uma grande seção, ficção ou não ficção, refina-se depois pelo gênero (história, ciência, biografia, etc) e segue-se por subcategorias até alcançar a prateleira do livro.

O algoritmo HNSW opera de maneira análoga, criando um gráfico de múltiplas camadas onde cada vetor é representado como um nó. A arquitetura é hierárquica: as camadas superiores contêm menos nós, porém com conexões abrangentes que permitem navegar rapidamente por grandes áreas do espaço multidimensional. À medida que a busca desce para as camadas inferiores, a quantidade de nós aumenta e as conexões tornam-se mais específicas, permitindo um refinamento preciso até o mais próximo possível do alvo, sem ser necessário comparar a busca com todos os dados existentes.



Dessa forma, o Qdrant evita o uso de força bruta e restringe, de forma eficiente, o espaço de busca em grandes conjuntos de dados.

Parâmetros do Índice HNSW

Parâmetro	Finalidade	Efeito
m	Links (conexões) por nó	Aumentar melhora recall; usa mais RAM e tempo de construção.
ef_construct	Candidatos verificados na inserção	Aumentar melhora a qualidade do grafo; torna a indexação mais lenta.
hnsw_ef	Candidatos verificados na busca	Aumentar melhora a recall; torna as consultas mais lentas.

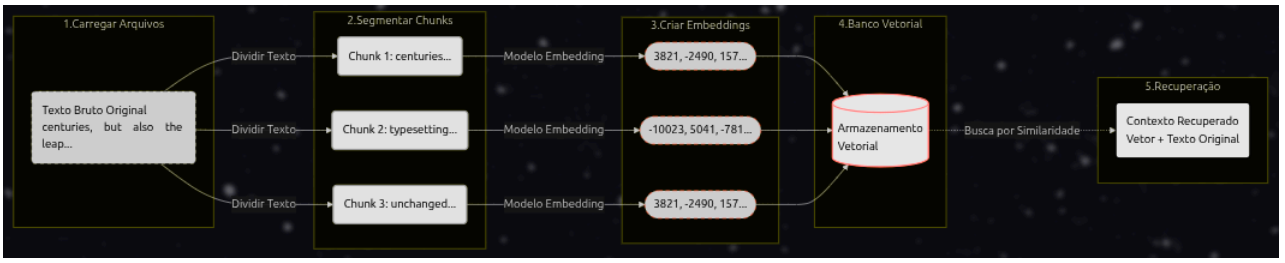
Esses conceitos sobre a plataforma Qdrant já são o suficiente para começar a implementar aplicações práticas. Após realizar parte do curso Qdrant Essentials, foi a vez de pôr em prática o mecanismo de busca semântica.

3. Criando um Chatbot com RAG com N8N e Qdrant

A arquitetura RAG (*Retrieval-Augmented Generation*) é usada para otimizar a precisão e a confiabilidade de modelos generativos, fundamentando suas respostas em dados externos, em vez de depender apenas do treinamento prévio do modelo. Seu fluxo de trabalho segue um ciclo de inserção, recuperação e geração.

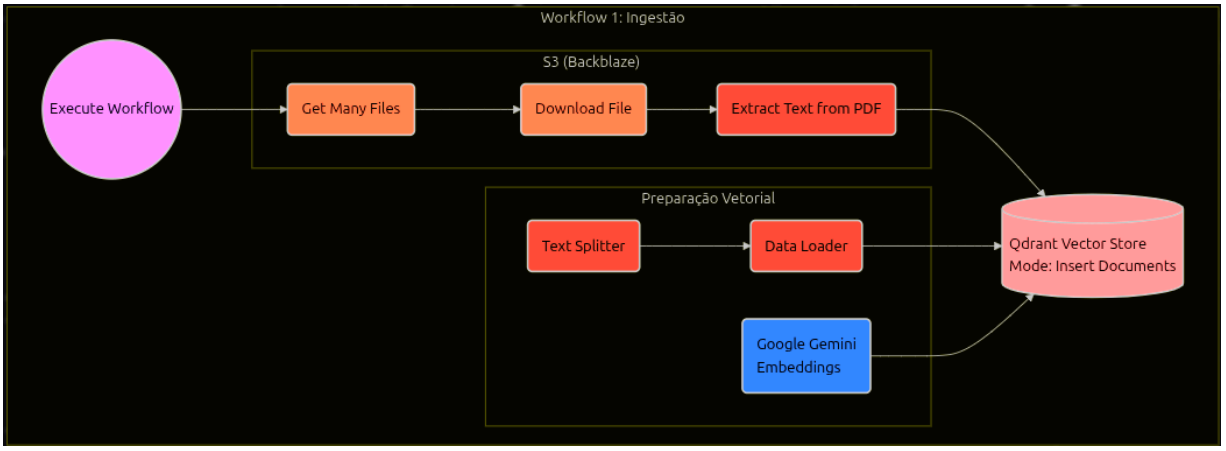
O fluxograma abaixo ilustra as etapas necessárias para se implementar a arquitetura RAG. O documento é inserido, seu texto é extraído, segmentado e transformado em embeddings, que são armazenados no banco vetorial para que o agente, ao precisar realizar consultas, possa recuperar informações para gerar uma resposta mais contextualizada.

Arquitetura RAG

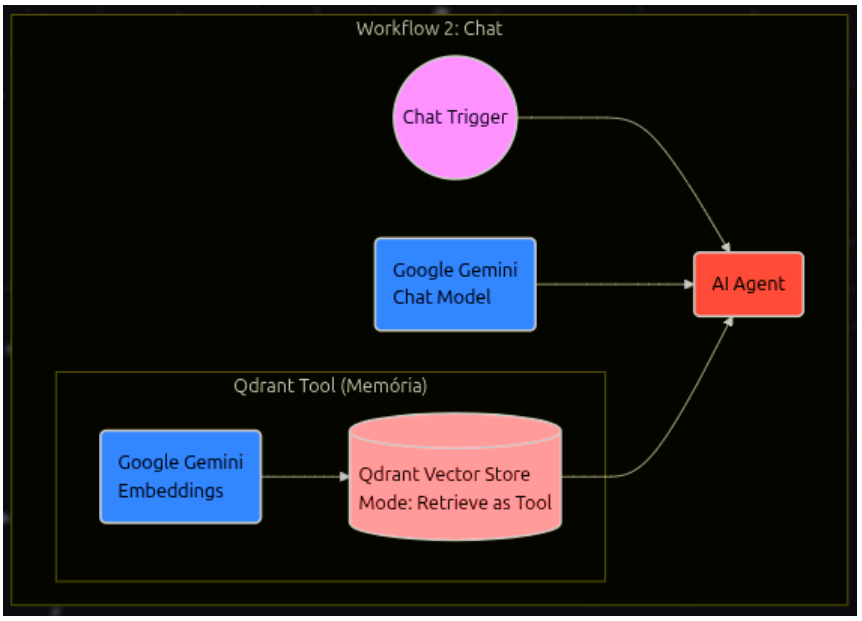


A arquitetura pode ser dividida em dois fluxos para facilitar a implementação, o fluxo de ingestão de dados onde os documentos e arquivos são transformados em embeddings e armazenados no banco vetorial, ficando disponíveis para o segundo fluxo, que é a recuperação desses dados por parte do agente.

Fluxo de Ingestão de Dados

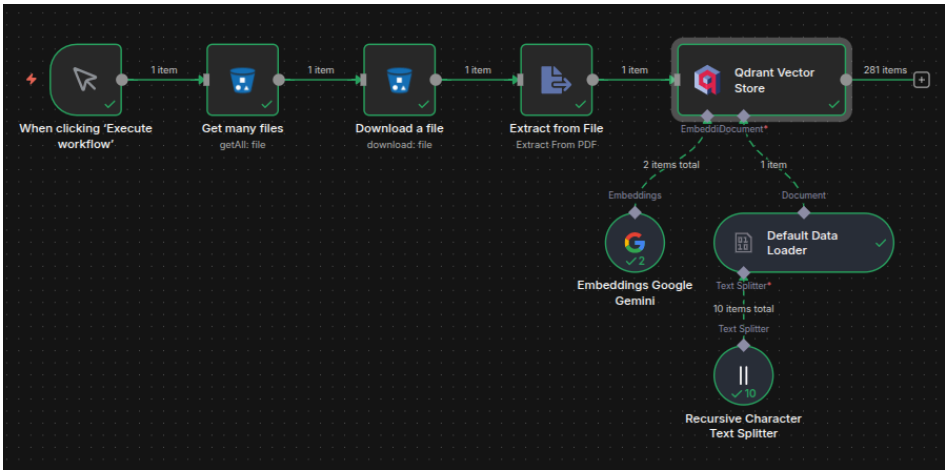


Fluxo de Recuperação pelo Agente

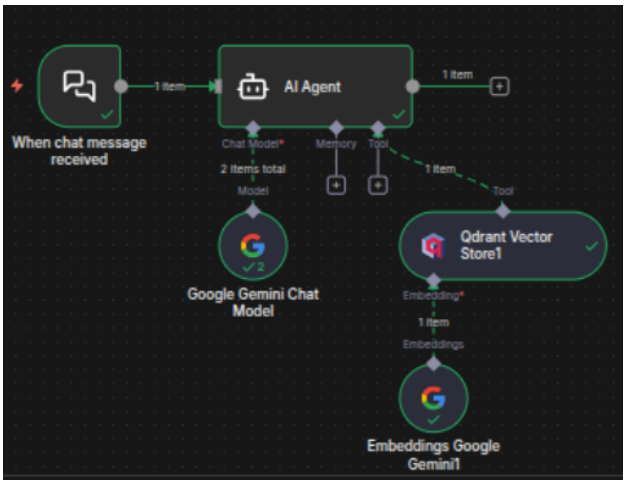


Entendida a arquitetura e suas etapas, é hora de implementar os fluxos de trabalho via N8N.

Fluxo de Ingestão Implementado no N8N

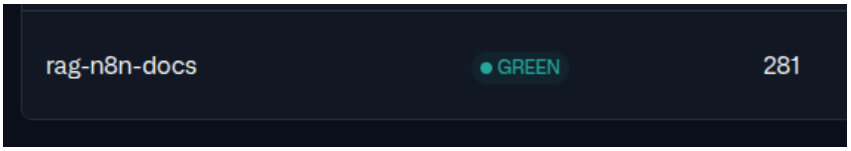


Fluxo de Recuperação Implementado no N8N

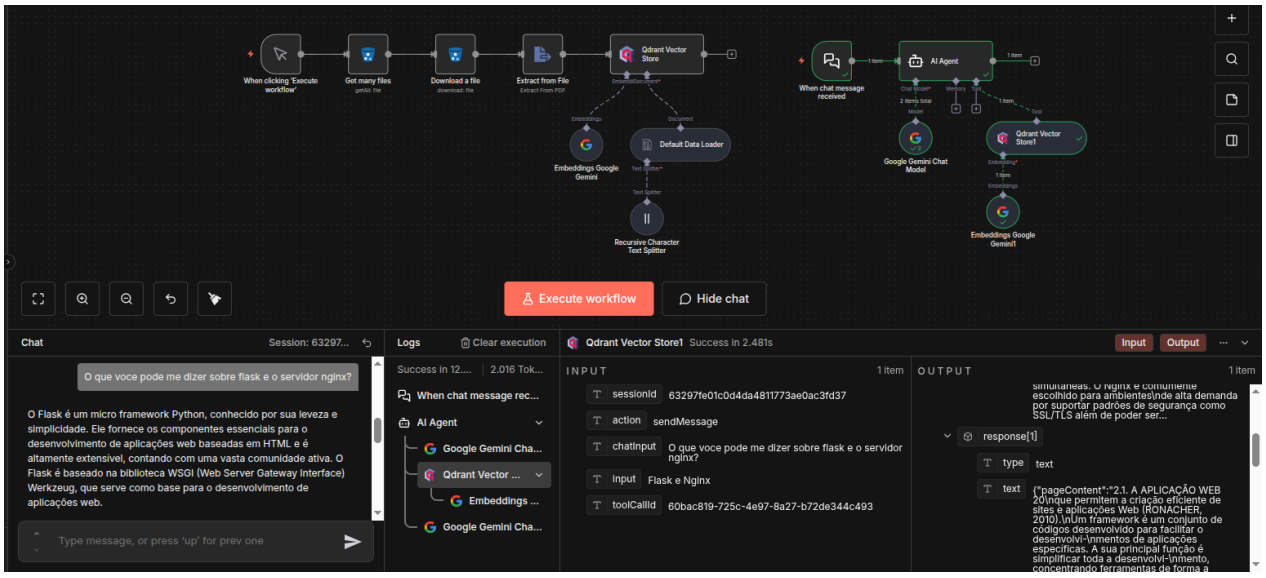


Usei meu TCC como contexto, ele possui informações sobre tecnologias como Nginx, Arduino, Esp32, MQTT, Flask, entre outros. Para utilizá-lo foi necessário transformá-lo em embeddings armazenados no Qdrant para que o agente tivesse acesso a essas informações No fluxo de recuperação do agente não foi adicionado nó de memória, exatamente para o modelo não usar informações não contidas no banco vetorial.

Coleção Criada no Cluster Qdrant Usando TCC como Referência



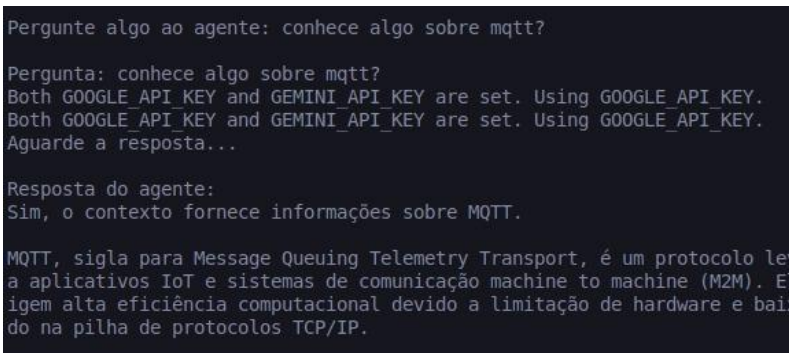
Arquitetura RAG em Execução no N8N



A implementação da arquitetura RAG via N8N validou a orquestração entre armazenamento de objetos, modelos de linguagem e busca vetorial. Conforme demonstrado nos testes, o sistema foi capaz de processar a ingestão de documentos e realizar a recuperação contextual precisa através do nó Qdrant Vector Store, respondendo corretamente a consultas específicas sobre frameworks e servidores web (como Flask e Nginx). Fica evidente que o uso de uma ferramenta de recuperação conectada ao Agente de IA diminui consideravelmente as alucinações, garantindo que o LLM (Google Gemini) mantenha suas respostas conectadas ao contexto indexado na base vetorial.

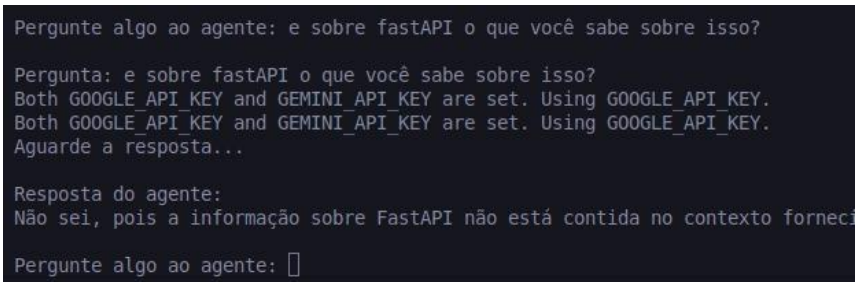
Para diminuir a abstração visual e ter outra forma de implementação sobre a manipulação vetorial, implementei também a lógica da arquitetura RAG em um ambiente de desenvolvimento utilizando Python e LangChain. Esta decisão foi impulsionada pelo progresso nos primeiros módulos do curso de certificação Qdrant Essentials, que utiliza a linguagem Python. A refatoração do fluxo para scripts Python permitiu maior entendimento sobre customização de parâmetros e integração entre sistemas.

Agente Recuperando Informações do Qdrant



Como esperado, ao ser questionado sobre um tópico que não está presente no banco de busca semântica, o modelo retorna uma resposta sincera, sem alucinações.

Agente Sem Informações Sobre o Contexto



Dificuldades

Sem maiores dificuldades

Conclusões

Por fim, o card consolidou a viabilidade da implementação de sistemas RAG utilizando uma arquitetura moderna baseada em Qdrant e Google Gemini. A estratégia de iniciar pela prototipagem visual no n8n para validação rápida de fluxos e posteriormente implementar via código em Python com LangChain facilitou o entendimento do processo de orquestração, além de ajuda a fixar conceitos estudados sobre busca vetorial e indexação vistos no curso Qdrant Essentials. O resultado final é uma solução funcional, capaz de entregar respostas contextualizadas e com menor risco de alucinação, estabelecendo a base para o desenvolvimento de agentes mais complexos

Referências

[1] QDRANT.TECH. **Qdrant Essentials Course**

Disponível em: <https://qdrant.tech/course/essentials/>

Acesso em 06 de janeiro de 2026

[2] YOUTUBE.COM. **Qdrant Essentials | Building Simple Vector Search in Qdrant**

Disponível em: <https://www.youtube.com/watch?v=_83L9ZIoOjM>

Acesso em 06 de janeiro de 2026

[3] YOUTUBE.COM. **RAG no N8N: Guia Definitivo. Retrieval-Augmented Generation Avançado com Arquivo Jurídico**

Disponível em: <https://www.youtube.com/watch?v=r74bYStawHU>

Acesso em 08 de janeiro de 2026

[4] QDRANT.TECH. **Cloud Quickstart - Qdrant**

Disponível em: <https://qdrant.tech/documentation/quickstart-cloud/>

Acesso em 06 de janeiro de 2026