Em direção a um Sistema de Busca com foco em relevância para o usuário em Strateegia

Pedro Meira-Betmann[1], Vinicius Cardoso Garcia[1]

[1]Centro De Informática, Universidade Federal de Pernambuco(UFPE), Recife, Brasil

{pm, vcg}@cin.ufpe.br

Abstract

This work aims to develop a search engine for the *Strateegia* platform by TDS.company, seeking to limit information overload as studied by Fasolo, Gigerenzer, and Goldstein. Utilizing the *Tractatus Framework*, we aim to implement an accessible search method that allows users to find the desired *templates* without prior knowledge, improving usability for lay users. The project aims to overcome the negative effects of excessive options by harmonizing the platform's complexity with the simplicity of the user interface, facilitating access to functionalities such as *templates* and *kits*.

Keywords: Search Engine, Recommendation System, Elasticsearch, Strateegia Platform, Large Language Models (LLMs), Tractatus Framework

Resumo

Este trabalho tem como objetivo desenvolver um engenho de busca para a plataforma *Strateegia* da TDS.company, visando limitar a sobrecarga de informação como estudado por Fasolo, Gigerenzer e Goldstein. Utilizando o *Framework Tractatus*, buscase implementar um método de busca acessível que permite aos usuários encontrarem as *templates* desejadas sem conhecimento prévio, melhorando a usabilidade para usuários leigos. O projeto visa superar os efeitos negativos do excesso de opções, harmonizando a complexidade da plataforma com a simplicidade da interface do usuário e facilitando o acesso a funcionalidades como *templates* e *kits*.

Keywords: Engenho de Busca, Sistema de recomendação, Elasticsearch, Strateegia Platform, Large Language Models (LLMs), Framework Tractatus

1 Introdução

A capacidade de não apenas seduzir, mas também estar acessível aos usuários quando estes buscam um produto qualquer, é crítica para o sucesso de qualquer artefato moderno, independente de se ele é físico, digital, ou uma combinação deles, sendo figital (Meira, 2021). Na era digital, a sobrecarga de escolhas devido à disponibilidade de informação quase infinita se mostra como uma barreira no processo de tomada de decisão (Fasolo et al., 2007). Esta questão da escolha também foi explorada por Gigerenzer and Goldstein (1996); Goldstein and Gigerenzer (2002) revelando que as pessoas, muitas vezes, se atraem por uma única característica marcante em vez de um conjunto delas. Tal conceito também se aplica à maneira como diferenciamos produtos ou informação, valorizando títulos em detrimento de textos menos destacados.

Com essas dificuldades em mente, engenhos de busca como o Google, idealizado por Page and Brin (1998), e o Radix (Fernandes, 2001), fruto de um projeto acadêmico na UFPE entre a última década do século XX e a primeira do século XXI, surgiram para facilitar o encontro entre a demanda do usuário e a oferta de informação na web. Posteriormente, observamos a criação de sistemas de recomendação, exemplificados pelo algoritmo de TikTok, que se assemelha ao Monolith de Liu et al. (2022), e pelo modelo de Amazon descrito por Linden et al. (2003), que se propõem a simplificar as opções do usuário, e o direcionar ao conteúdo que a plataforma acredita que o cliente irá preferir. Desse modo, engenhos de busca tem controle maior sobre as respostas e sistemas de recomendação tem uma simplicidade maior de uso, contudo, para um sistema completo, é necessário ambos (Lee et al., 2007).

Uma solução que combina controle e simplicidade é a criação de assistentes com inteligência artificial com apoio de frameworks estruturados, como o Tractatus e o Graph-RAG. O Tractatus, apresentado por Neves et al. (2024), baseia-se na obra filosófica de Wittgenstein (1922) para implementar uma estrutura hierárquica em JSON, que organiza o conhecimento de forma lógica e acessível para o treinamento de Large Language Models (LLMs) e técnicas de Retrieval-Augmented Generation (RAG). Essa abordagem permite uma representação clara e modular de informação, facilitando a integração e a atualização dinâmica do conhecimento existente nos modelos. Em contraste, o Graph-RAG, conforme descrito por Peng et al. (2024), representa uma evolução dos modelos RAG tradicionais ao incorporar grafos de conhecimento que capturam relações semânticas mais complexas entre os dados. Enquanto o Tractatus foca na hierarquização, clareza estrutural e simplicidade por meio de JSON, o Graph-RAG expande essa estrutura ao utilizar grafos para mapear interconexões detalhadas sendo mais robusto e por sua vez necessitando mais poder computacional.

Com base nos estudos apresentados na introdução, é oportuno analisar como essas teorias se aplicam em um contexto prático. Nesse sentido, a plataforma *Strateegia*, descrita por Neves et al. (2020), com a qual temos proximidade pelo ambiente de trabalho, surge como um caso relevante para examinar a implementação de tecnologias avançadas em ambientes de inovação, especialmente por se destacar pelo compromisso em adotar técnicas de ponta, apresentando-se como um ambiente robusto para inovações.

No entanto, ao analisar detalhadamente a plataforma, especialmente sua funcionalidade de *templates*, identifica-se falhas significativas em seu sistema de busca atual. O sistema não

aproveita plenamente a riqueza de informação disponíveis nem implementa técnicas avançadas, como utilização de assistentes LLM contextualizados com informação relevante ou algoritmos de recuperação otimizados, como a lógica fuzzy (Zadeh, 1965), resultando num engenho de busca que não retorna a informação requisitada pelo usuário de maneira eficiente. Essas deficiências se concretizam em uma sobrecarga informacional para os usuários, dificultando a localização precisa de conteúdos pertinentes e refletindo os desafios mencionados anteriormente.

Para mitigar esses problemas, propõe-se a implementação de um mecanismo de busca e recomendação capaz de permitir ao usuário buscar conteúdos de forma mais precisa e menos rigorosa. Utilizando, por exemplo, técnicas como o *Tractatus*, é possível viabilizar buscas semânticas e recomendações de conteúdo alinhadas às necessidades explicitadas pelo usuário. Essa abordagem visa atualizar o sistema de busca, garantindo que os resultados reflitam com maior exatidão a intenção original do usuário. A combinação das teorias discutidas com sua aplicação prática em *Strateegia* revela um caminho promissor para o desenvolvimento de um sistema que equilibre, de maneira eficaz, a complexidade tecnológica com o controle do usuário, essencial para o crescimento sustentável e a evolução contínua de qualquer produto digital.

O objetivo geral deste trabalho é explorar e implementar técnicas de busca com o apoio de LLMs para aprimorar o processo de busca para o usuário, tentando eliminar a barreira de conhecimento, resultando nos objetivos específicos seguintes.

1. **Substituição do sistema atual de busca de *Strateegia* por um que utilize todos os campos disponíveis e tenha tolerância de erros do usuário**
2. **Criação e teste de um assistente feito com apoio de um *Framework Tractatus* com dados de *Strateegia***
3. **Utilização desse *Tractatus* para melhorar a experiência de busca para usuários leigos**

2 Fundamentação teórica e trabalhos relevantes

Antes de explicar a metodologia e implementação das soluções propostas neste trabalho, é necessário compreender os conceitos que guiaram esse trabalho e as ferramentas utilizadas.

2.1 Strateegia

O *Strateegia* (tds.company, 2024) é uma plataforma ágil e colaborativa projetada para facilitar a cocriação, o gerenciamento estratégico de projetos e o aprendizado coletivo. A plataforma utiliza diferentes pontos para alcançar esses objetivos, os primeiros e mais fundamentais são os pontos de divergência e convergência que compõem o *Double Diamond*, ideia inicialmente desenvolvida por Banathy (1996) e popularizada pelo conselho britânico de design, como foi descrito em Design Council (2007). Em *Strateegia* os pontos de divergência são utilizados para discutir ideias e criar possibilidades, em contrapartida os de convergência são utilizados para votar e tomar decisões sob os temas discutidos anteriormente. Além desses pontos, existem

também pontos de aviso para dar notícias, de conversação para marcar uma reunião, avaliação para corrigir respostas, e monitoramento para avaliar a performance da jornada.

Esses pontos ficam em mapas. Mapas podem ser compreendidos como representações do tópico geral de uma discussão ou técnica a ser utilizada pela equipe, se tornando o espaço para adicionar os pontos desejados.

Da mesma maneira que mapas contém pontos, jornadas contém mapas. Jornadas são o ambiente coletivo de discussão, a ser usado a longo prazo, contendo mapas e usuários individuais.

É possível criar tanto mapas como jornadas utilizando *templates* disponíveis em *Strateegia*. Ao utilizar um *template*, o usuário replica um mapa ou jornada previamente publicado pela plataforma ou pela comunidade.

Com o objetivo de facilitar a compreensão do usuário tanto de como utilizar os pontos e técnicas propostas como também do tema sendo discutido, esses pontos contém links complementares adicionados pelos próprios usuários, referências e uma introdução que são adicionadas no processo de criação do ponto.

Como resultado das últimas evoluções na plataforma, *Strateegia* incorporou o uso de inteligência artificial para apoiar os seus usuários, esse apoio acontece de algumas maneiras diferentes como a criação de assistentes para apoio do usuário, geração de relatórios e resumos, monitoramento de performance do grupo e até avaliações de respostas.

2.2 Templates Strateegia

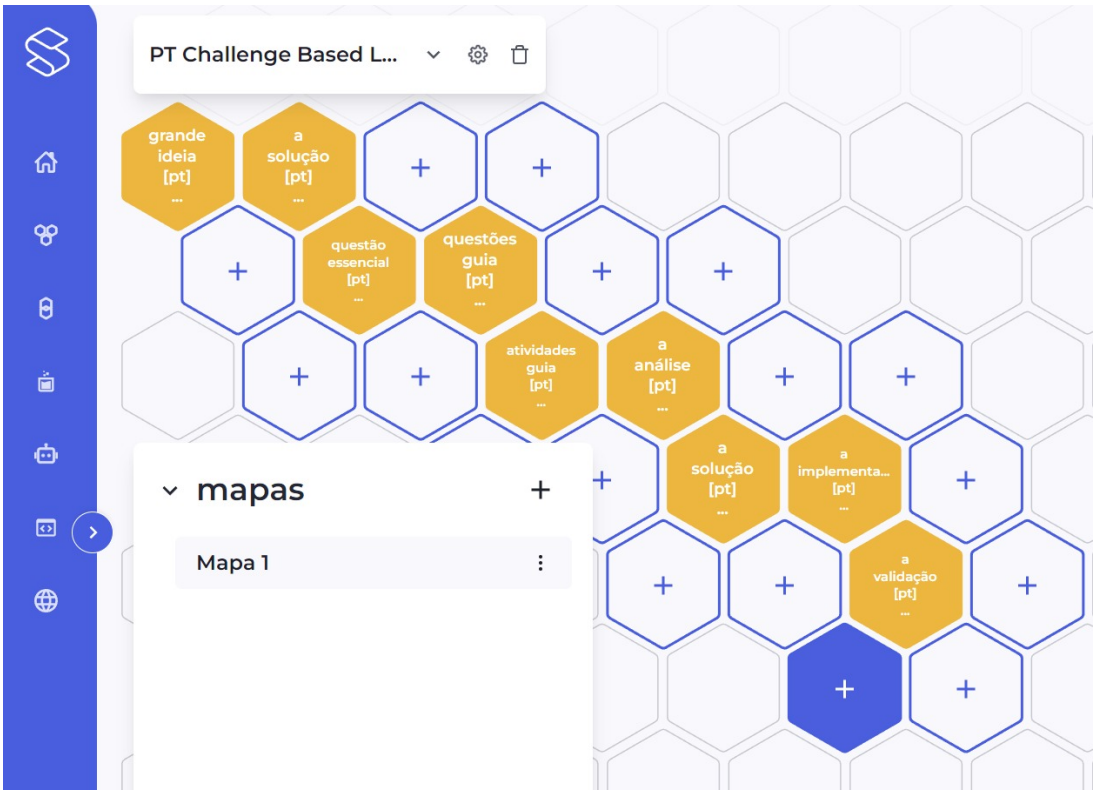


Figura 1: Mapa 1 da jornada PT Challenge Based Learning em strateegia

A jornada representada na figura 1 foi criada utilizando o *template* denominada PT Challenge Based Learning (CBL); este *template* tenta replicar a técnica de CBL utilizando diversos pontos de divergência para gerar questões sobre o tema, atividades e validando as soluções criadas.

A utilização de *templates* pode ser uma ferramenta poderosa que possibilita a criação rápida e eficiente de mapas e jornadas que aplicam diversas técnicas de aprendizado e tomada de decisão, incluindo de áreas como marketing e design.

Apesar de seu potencial, os *Templates* são pouco usados na plataforma devido à sua alta complexidade. Existem centenas de *templates* na plataforma, cada uma delas implementando uma técnica específica desenvolvida para um momento particular de um projeto. Essa complexidade pode gerar sobrecarga de informação em usuários desprovidos do conhecimento das estratégias replicadas.

As *Templates* são projetadas para ajudar o usuário e guiá-lo por um caminho que o leve a tomar decisões mais informadas, ou descobrir melhores perguntas a serem feitas sobre seus projetos. Contudo, independente da qualidade e exatidão que a técnica foi replicada, esses pontos positivos são subutilizados quando o usuário não tem a capacidade de encontrar, ou escolher, a técnica que ele deseja utilizar na plataforma.

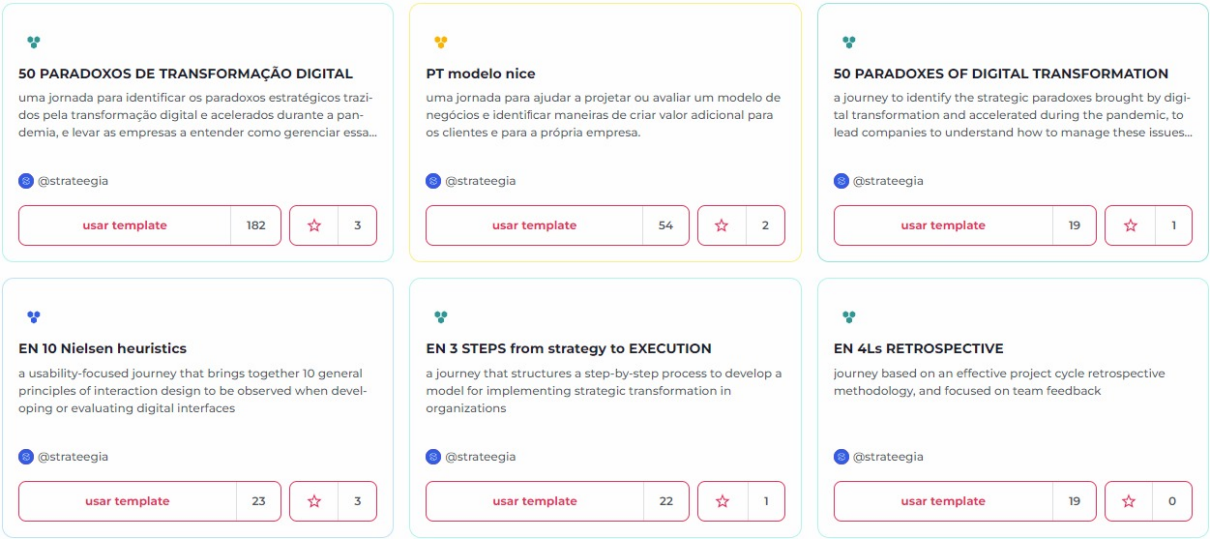


Figura 2: Pagina de seleção de templates em Strateegia

2.3 Sobrecarga de informação

A sobrecarga de informação refere-se ao fenômeno em que a quantidade de dados ou opções disponíveis excede a capacidade de processamento do indivíduo, levando à dificuldade na tomada de decisões. Esse excesso de informação pode resultar em indecisão, estresse e insatisfação, uma vez que os consumidores se sentem incapazes de analisar todas as alternativas de forma eficaz.

Em Fasolo et al. (2007), os autores discutem como, na era da internet, os consumidores enfrentam a tirania da escolha devido à disponibilidade quase infinita de produtos e informação. Eles argumentam que essa abundância pode ser evitada se os consumidores focaram em atributos específicos dos produtos, especialmente quando esses atributos não estão em conflito ou

possuem diferentes níveis de importância. Nessa perspectiva, mesmo ignorando a maior parte da informação, é possível fazer escolhas satisfatórias baseando-se em um único atributo relevante, mitigando assim os efeitos da sobrecarga de informação.

Em Gigerenzer and Goldstein (1996) os autores exploram como os organismos fazem inferências indutivas utilizando heurísticas simples e eficientes, conhecidas como heurísticas "rápidas e frugais". Eles sugerem que, em vez de processar toda a informação disponível, os indivíduos utilizam estratégias cognitivas que requerem menos recursos e tempo, permitindo decisões eficazes em ambientes complexos. Essa abordagem reconhece a sobrecarga de informação como um problema e propõe que a simplicidade nas estratégias de decisão pode ser uma solução adaptativa para lidar com o excesso de dados.

Analisando esses estudos, a importância da simplicidade ao demonstrar opções fica clara, especialmente numa plataforma que se propõe a fazer algo inovador como *Strateegia*. Analisando a plataforma pela quantidade de ações que você pode fazer ela parece muito simples, mas por ser tão simples ela tem possibilidades infinitas e pode ficar muito complicada para o usuário leigo.

A melhora da busca na plataforma e, principalmente, a implementação de um assistente que ajuda o usuário a tomar decisões de maneira mais fácil, levando em conta as suas principais dores, tenta limitar as dificuldades trazidas pelo desconhecido. Fazendo isso podemos explicitar a simplicidade da plataforma: caso o usuário queira alcançar um objetivo qualquer, tudo que ele precisa fazer é seguir o processo descrito nas templates.

2.4 Arquitetura da Plataforma

O back-end da plataforma, responsável pela captura, processamento, armazenamento, recuperação e distribuição de informação, foi desenvolvido utilizando a linguagem de programação *Kotlin*. *Kotlin* é uma linguagem moderna e estaticamente tipada que opera na Java Virtual Machine (JVM), permitindo total interoperabilidade com Java e facilitando o uso do *Spring Framework*. O *Spring Framework* é um framework de código aberto amplamente utilizado para o desenvolvimento de aplicações empresariais, oferecendo suporte a padrões de projeto como injeção de dependências e programação orientada a aspectos.

Todo o armazenamento é realizado utilizando o *MongoDB*, um banco de dados NoSQL orientado a documentos que utiliza esquemas dinâmicos, o que facilita a integração de dados em aplicações que requerem flexibilidade e escalabilidade. A infraestrutura adotada é baseada em microsserviços, sendo o serviço utilizado neste projeto o *template service*, representado como *sp-templates* no *MongoDB*, que lida com o armazenamento de *templates* de mapas e jornadas e *templates* de pontos.

2.5 Modelos de Linguagem de Grande Escala (LLMs)

Modelos de linguagem de grande escala (LLMs) são modelos de inteligência artificial treinados em enormes conjuntos de dados textuais, capazes de compreender e gerar linguagem natural de forma coerente (Naveed et al., 2023). Eles são utilizados em diversas aplicações, como assistentes

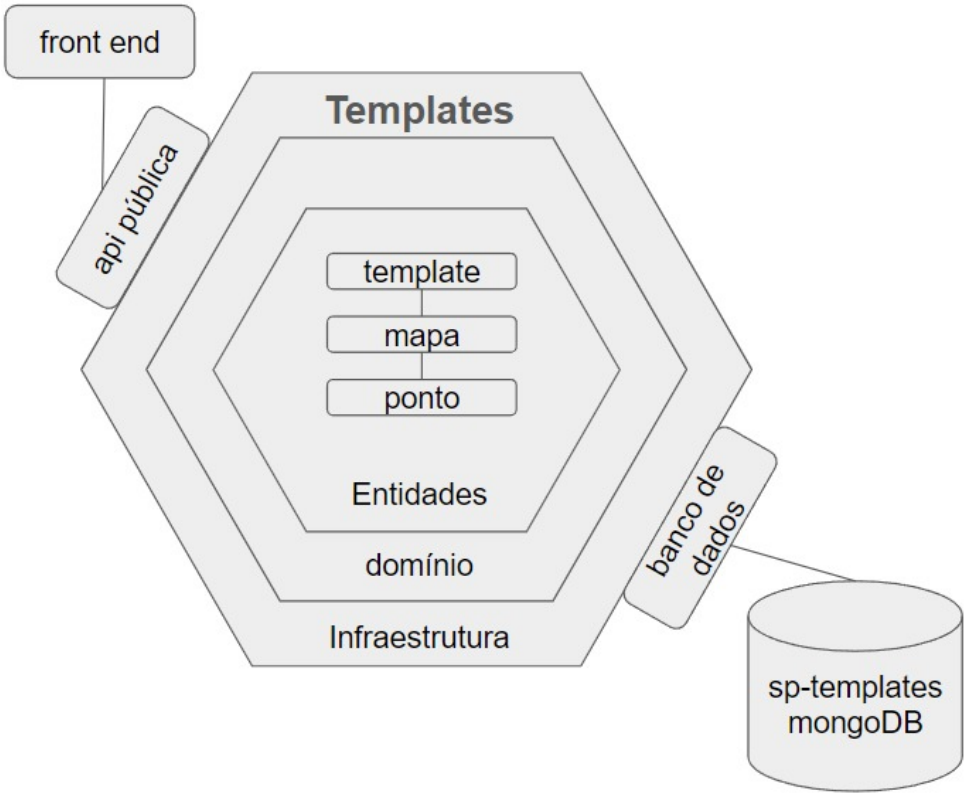


Figura 3: Arquitetura do microserviço de Templates

virtuais, tradução automática e análise de sentimentos. No projeto atual, os LLMs são empregados para aprimorar a capacidade do mecanismo de busca em interpretar consultas complexas e fornecer respostas contextualizadas, tornando a interação com o sistema mais natural e eficiente para o usuário.

Atualmente, um dos usos principais de LLMs é a criação de assistentes virtuais treinados e contextualizados do problema que o usuário enfrenta. A criação de um assistente pode ser feita de diversas maneiras, como a adição de contexto complexo (Peng et al., 2024) ou criando responsabilidades que ele precisa seguir e determinando a resposta correta utilizando exemplos (Dong et al., 2024). Tais assistentes podem ser explorados de diversas maneiras dependendo unicamente do contexto adicionado. Neste artigo utilizaremos assistentes treinados com o contexto de *Templates* para apoiar usuários no processo de busca da solução mais relevante baseada nas suas necessidades.

2.6 Tractatus

O *Framework Tractatus* é inspirado na hierarquia lógica de proposições de Wittgenstein (1922). Ele emprega o *JavaScript Object Notation* (JSON) para representar proposições primárias e secundárias de maneira estruturada e hierárquica. O JSON é um formato de dados leve e flexível, facilmente legível por humanos e máquinas.

O objetivo principal do *Framework Tractatus* é ampliar a arquitetura fundamental dos Modelos de Linguagem de Grande Escala (LLMs) ao fornecer uma base de conhecimento estruturada

e hierárquica. Este *framework* foi escolhido especificamente por seu potencial de mesclar a base de conhecimento pré-treinada dos LLMs com técnicas de *Retrieval-Augmented Generation* (RAG) (Peng et al., 2024) e pela proximidade com seu criador. Sua utilização permite uma combinação eficaz entre o conhecimento existente do modelo e a informação recém-integrada. Isso é alcançado através da estratificação lógica da informação, que não apenas facilita uma compreensão profunda de conceitos complexos, mas também permite a integração dinâmica de nova informação. Ao aproveitar a simplicidade e a versatilidade inerentes do JSON, o *framework* garante que os dados organizados permaneçam acessíveis e interpretáveis pelos LLMs, aprimorando suas capacidades de processamento e geração de texto semelhante ao humano.

Ademais, a elaboração do *tractatus* é conduzida de maneira semântica e não sintática. Dessa forma, ao sintetizar a informação, apenas os elementos relevantes são armazenados em proposições, podendo estas constituir parte ou a totalidade do texto.

Na figura 3, demonstramos a estrutura do *Tractatus*, uma proposição principal e sub proposições qualificando ela, deixando o seu objetivo mais claro e específico para melhor compreensão do leitor e da LLM. Sua lógica pode ser ampliada, adicionando relações entre proposições, títulos entre outras propriedades que deixam as relações mais completas e explícitas.

O *Tractatus* desenvolvido para este projeto foi construído utilizando os dados extraídos da base de dados de *templates* da plataforma *Strateegia*. Durante sua implementação, observamos que a sofisticada capacidade de compreensão contextual das LLMs permitiu algumas simplificações estruturais importantes. A primeira delas foi a possibilidade de unificar título e proposição principal em um único campo, sem prejuízo ao entendimento do modelo. De maneira similar, embora seja possível estruturar as proposições secundárias em um array específico (*subpropositions[]*), verificamos que as LLMs são capazes de compreender naturalmente a hierarquia numérica (por exemplo, que a proposição 1.1 está subordinada à proposição 1), tornando desnecessária essa estruturação adicional.

O *Tractatus* tem uma estrutura flexível. Assim como é possível adicionar títulos, também podem ser adicionadas descrições extras, criar relacionamentos complexos adicionando links para outras proposições e até links externos, caso seja necessário que o assistente retorne essa informação.

Analisando a figura 4, sua primeira proposição descreve o *template* de PT Business model, a representação em *Strateegia* da ferramenta de gerenciamento estratégico *Business Model Canvas*, desenvolvida originalmente por Osterwalder (2008). Enquanto sua proposição principal descreve o objetivo geral do *template*, suas proposições secundárias descrevem o que está contido nela. Neste caso seus componentes chave são os 9 pontos discutidos dentro de um *Business Model Canvas*.

Assim, ao incorporar esse documento como parte do contexto de um LLM, há uma redução significativa no risco de geração de informação imprecisa ou "alucinações" (Peng et al., 2024), aumentando a probabilidade de que o modelo retorne informação relevante às solicitações.

```
{
  "1": {
    "proposition": "PT␣Business␣Model␣provides␣a␣practical␣and␣
       agile␣method␣for␣developing␣business␣models.",
    "1.1": {
      "proposition": "Key␣components␣include␣the␣value␣
         proposition,␣resources,␣cost␣structure,␣revenue␣structure
         ,␣customer␣segments,␣relationships,␣and␣channels."
    }
  }
}
```

Figura 4: Parte de um Tractatus gerado pelo assistente utilizando dados de templates Strateegia

2.7 RAG e GraphRAG

No artigo *Graph Retrieval-Augmented Generation: A Survey* (Peng et al., 2024) os autores revisam metodologias de Geração Aumentada por Recuperação(RAG) focadas em grafos, chamadas GraphRAG. Os autores destacam que, embora a RAG melhore modelos de linguagem (LLMs) sem re-treinamento, enfrenta limitações ao lidar com relações complexas entre entidades. GraphRAG é proposto para superar essas limitações, utilizando informação estrutural de grafos para permitir recuperações mais precisas e respostas contextuais aprimoradas.

Este artigo é relevante para o presente estudo porque demonstra como a RAG pode aprimorar assistentes virtuais, permitindo que LLMs acessem conhecimento externo e mitigando problemas como alucinações e falta de informação atualizada. GraphRAG amplia essa abordagem ao incorporar grafos, capturando relações complexas entre entidades.

O *Framework Tractatus* utiliza uma hierarquia lógica implementada em JSON para representar proposições e subproposições de maneira estruturada e estratificada. Essa hierarquia permite uma integração eficiente com LLMs, facilitando a compreensão e a geração de texto de forma lógica e sequencial, com menor demanda computacional. Por outro lado, GraphRAG emprega estruturas de grafos para modelar relações complexas e interconectadas entre entidades, permitindo uma recuperação de informação mais dinâmica e flexível, mas que pode exigir maior capacidade computacional. Enquanto o *Tractatus* se destaca pela simplicidade e eficiência na organização lógica do conhecimento, *GraphRAG* oferece profundidade na captura de interconexões complexas. Assim, a escolha entre uma estrutura hierárquica e uma de rede depende das necessidades específicas do sistema, podendo ambas as abordagens ser complementares na melhoria de mecanismos de busca e assistentes inteligentes.

Para este projeto, a proximidade com o desenvolvedor do *framework Tractatus* e sua simplicidade, tanto na criação quanto na utilização, foram fatores determinantes na decisão de adotá-lo no lugar de *GraphRAG*.

2.8 Sistemas de recomendação

Um sistema de recomendação é uma ferramenta que analisa dados de usuários e itens para fornecer sugestões personalizadas, auxiliando os usuários a encontrar produtos ou conteúdos de seu interesse em meio a um vasto conjunto de opções. Esses sistemas utilizam algoritmos que identificam padrões de comportamento e preferências, aprimorando a experiência do usuário ao oferecer recomendações relevantes e precisas.

No artigo *Monolith: Real Time Recommendation System With Collisionless Embedding Table* (Liu et al., 2022), os autores apresentam um sistema de recomendação em tempo real que utiliza uma tabela de embeddings sem colisões. O sistema, denominado Monolith, foi projetado para ser escalável e interagir com feedbacks dos usuários em tempo real, melhorando a qualidade das recomendações em aplicações como classificação de vídeos curtos e anúncios online.

Por sua vez, (Linden et al., 2003), no artigo *Amazon.com Recommendations: Item-to-Item Collaborative Filtering*, descrevem o uso de filtragem colaborativa baseada em itens para gerar recomendações personalizadas na Amazon.com. Esse método foca em encontrar similaridades entre itens, ao invés de usuários, permitindo que o sistema seja altamente escalável e forneça recomendações de alta qualidade em tempo real, mesmo diante de um grande volume de dados e usuários.

Nesta pesquisa utilizamos uma espécie de sistema de recomendação, diferentemente desses outros projetos o nosso não é automático e precisa de input de usuário, utilizando o *Tractatus* podemos criar contexto adicional utilizando algum LLM, como o ChatGPT desenvolvido pela *OpenAI*, e nessa conversa específica, ele irá aprender toda informação contida neste documento e servirá de sistema de recomendação para o usuário. Numa conversa com ele é possível perguntar qual a melhor maneira de fazer uma ação qualquer e ele me responderá dentro do seu contexto, nesse caso, uma recomendação de qual *template* eu posso utilizar que melhor me serviria.

2.9 Mecanismo de Busca e Pagerank

Um mecanismo de busca é um sistema projetado para realizar buscas em um conjunto de dados, retornando informação relevante com base em consultas dos usuários (Baeza-Yates & Ribeiro-Neto, 1999). Ele utiliza algoritmos para indexar dados, processar consultas e classificar os resultados de acordo com a relevância. Exemplos conhecidos incluem Google (Page & Brin, 1998).

No contexto desse projeto, a compreensão do mecanismo utilizado por Google é importante para entender as decisões tomadas. PageRank (Page et al., 1999) baseia-se na análise da estrutura de links entre documentos para determinar a autoridade e a importância de uma página. Em contraste, nossa função de busca opera em uma coleção de *templates* que não possuem qualquer noção de interligação ou referências entre si. Nosso foco é na relevância baseada no conteúdo, correspondendo os termos da consulta aos textos dentro de cada *template*.

Além disso, o PageRank fornece um ranqueamento independente da consulta, ou seja, a pontuação de PageRank de uma página não muda com base na consulta do usuário. Em nosso caso, o ranqueamento é dependente da consulta, calculando pontuações com base em quão bem cada

template corresponde aos termos específicos da consulta do usuário. Nossa prioridade é atender à intenção do usuário em uma consulta específica, em vez de apresentar um ranqueamento geral de importância.

2.10 Elasticsearch

Elasticsearch é uma tecnologia de código aberto para armazenamento, busca e análise de grandes volumes de dados em tempo real. Baseado na biblioteca Apache Lucene, ele oferece um mecanismo de busca distribuído, permitindo indexação e recuperação eficientes de dados em formato JSON. Como afirmado por Zamfir et al. (2019), o que o torna adequado para aplicações que exigem processamento rápido de consultas complexas, filtragem e agregação de dados. Sua arquitetura distribuída permite escalabilidade horizontal, tornando-o ideal para monitoramento de sistemas, análise de logs e outras aplicações que lidam com grandes quantidades de dados.

Uma característica importante do Elasticsearch é sua natureza como sistema independente, operando externamente à base de dados principal da aplicação. Esta arquitetura, embora poderosa, requer a implementação de um processo contínuo de sincronização para manter o índice de busca atualizado com os dados originais. Considerando que implementações alternativas, como buscas diretas no MongoDB, não apresentam essa necessidade de sincronização, optamos por utilizar o Elasticsearch exclusivamente como ferramenta de validação e comparação neste projeto. A implementação foi realizada em Python, escolha motivada pela simplicidade da linguagem e sua capacidade de prototipação rápida, permitindo um desenvolvimento ágil dos testes necessários.

3 Requisitos para a Busca

O projeto de implementação do mecanismo de busca para a plataforma *Strateegia* necessita de requisitos claramente definidos para alcançar seus objetivos. Adaptando os requisitos básicos para uma busca apresentados por Garcia et al. (2006), definimos as exigências a seguir para este caso:

a. Alta precisão e recall. É essencial que o sistema de busca ofereça alta precisão, garantindo que a maioria dos resultados retornados sejam relevantes para a consulta do usuário. Além disso, o recall elevado assegura que poucos resultados relevantes sejam omissos.

b. Formulação simplificada de consultas. Para atender às necessidades de usuários leigos, o mecanismo de busca deve permitir a formulação de consultas de maneira simples e intuitiva, tolerando erros e apoiando o usuário onde existirem lacunas de conhecimento.

c. Descrição detalhada de objetos e relevância dos campos. O engenho de busca deve ser capaz de interpretar e buscar informação em diferentes campos, como título, descrição e títulos dos mapas das *templates*.

d. Controle de privacidade. Embora o sistema de busca deva eventualmente respeitar os níveis de privacidade das *templates* (público ou privado), no contexto de testes, a função de busca foi implementada para ignorar temporariamente essas restrições, retornando todas os

templates independentemente do status de privacidade. Quando estiver em produção a função de busca respeitará essas necessidades.

e. Repositório e atualização automática de *templates*. Como o projeto envolve um sistema dinâmico, com constantes inserções de novos *templates*, é importante que o mecanismo de busca seja capaz de atualizar automaticamente seu índice para garantir que as buscas considerem a informação mais recente. No entanto, dado que o projeto é uma prova de conceito, optou-se por não incluir uma *pipeline* de atualização contínua de dados no MongoDB para o Elasticsearch. Este ponto também beneficia a implementação em kotlin compreendendo que esta implementação seria atualizada automaticamente sem necessidade de implementar uma pipeline de dados adicional.

f. Desempenho. Originalmente o desempenho seria medido utilizando tempo de retorno da busca, e num engenho de busca comum é algo extremamente importante. Apesar disso, no caso deste trabalho a velocidade em que o retorno é dado é muito menos importante do que a qualidade do retorno, a utilização de um assistente para apoiar você na decisão do *template* representa um aumento colossal em comparação ao tempo original de retorno. Por causa desse fator o tempo de busca geral é ignorado a favor de uma resposta melhor ao usuário.

4 Design, Projeto e Implementação da Proposta

A metodologia adotada na implementação deste projeto foi estruturada em etapas sequenciais e interdependentes, cada uma delas essenciais para o desenvolvimento e avaliação do mecanismo de busca proposto. As subseções a seguir detalham cada uma dessas etapas.

4.1 Seleção e Pré-Processamento de Dados

Inicialmente, utilizamos apenas a base de dados da plataforma *Strateegia*. Começamos os testes utilizando essa base, que continha 337 objetos referentes aos *templates*. Porém, ao longo do processo de análise da base, tornou-se evidente que ela não era adequada para demonstrar a qualidade da função de busca implementada. Isso se deve ao fato de a base estar extremamente contaminada com informação de teste que não poderiam ser facilmente filtradas ou limpas, e caso fossem ela não teria um tamanho adequado, além disso pode servir como um teste de caso extremo.

Diante dessa adversidade, optamos por selecionar uma segunda base de dados para realizar testes mais eficazes e precisos. A base alternativa selecionada foi a `fetch_20newsgroups` pelo seu texto se assimilar as descrições dos templates de strateegia, as categorias aos títulos e pela repetição de palavras entre diferentes objetos simulando a repetição existente nos templates.

Quanto à sua indexação, após os objetos incompatíveis com o *Elasticsearch* serem eliminados, a base continha 18.261 objetos, permitindo uma avaliação mais robusta das funcionalidades implementadas. Essa base mais ampla e limpa foi essencial para testar a eficácia real do mecanismo de busca sem interferências provenientes de dados inconsistentes.

Para a criação dos índices de Elastic feitos com a base de dados *Strateegia*, todos os objetos foram transformados em Strings para que não houvesse conflito com as funções já existentes do

Elastic, utilizamos os IDs dos mapas dentro dos *templates* existentes na base de dados para, acessando outra coleção, selecionar todos os pontos nos mapas do *template* salvando os seus títulos e descrições em um array de objetos no maior objeto *template*, isso iria ser salvo no Elastic com objetivo de possibilitar uma busca mais robusta, mas décimos por utilizar apenas o objeto original para termos uma comparação mais direta das funções.

Os dados utilizados para Kotlin foram clonados da coleção de *templates* do ambiente de desenvolvimento de *Strateegia*, utilizando uma instância de mongoDB rodando no Docker.

4.2 Desenvolvimento do Código em Python

O código inicial foi desenvolvido em Python, com a busca implementada utilizando o Elasticsearch. A escolha pelo Elasticsearch, em vez de criar uma função própria de busca, foi uma decisão estratégica. Considerando que o projeto seria implementado em uma plataforma funcional, o uso de uma ferramenta já consolidada e amplamente testada garantiu maior confiabilidade e eficiência. Embora tenhamos a habilidade de criar um mecanismo de busca do zero, o resultado não seria ideal em comparação com funções existentes que passaram por extensos testes e otimizações no mundo real. Outro fator é que o objetivo principal dessa implementação é ser comparada com a implementação em kotlin para garantir a sua qualidade

Implementamos técnicas que permitiram maior tolerância a erros de digitação utilizando lógica fuzzy (Zadeh, 1965) e ampliamos os critérios de busca para incluir não apenas títulos, mas também descrições e títulos dos mapas dos *templates*. Além disso, ajustamos os pesos de diferentes campos, atribuindo maior importância ao título em relação à descrição, o que aprimorou a relevância dos resultados retornados.

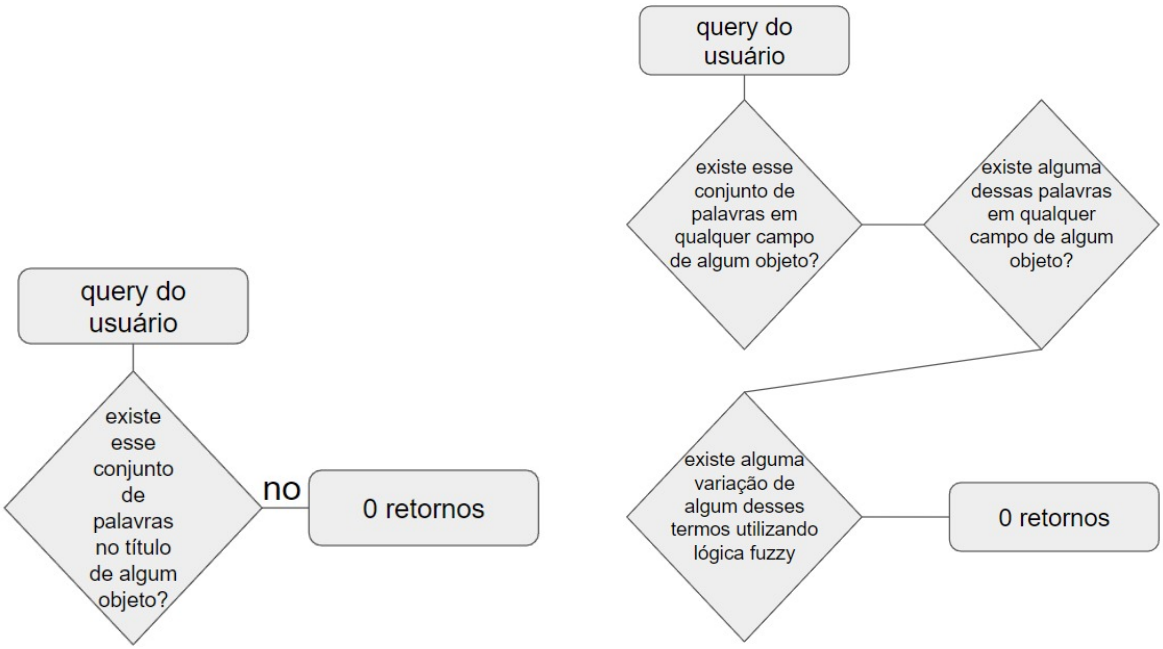
4.3 Implementação em Kotlin e Integração com MongoDB

Após a fase inicial de desenvolvimento em Python, desenvolvemos uma segunda versão do mecanismo de busca, desta vez implementada em Kotlin e conectada diretamente à base de dados MongoDB. A técnica aplicada espelhava a implementação realizada com o Elasticsearch, visando manter um padrão de qualidade e permitir uma comparação justa entre as duas abordagens.

Identificamos, entretanto, que a estrutura do objeto existente na base de dados não era adequada para a busca por diversas razões e será refeita no futuro. Com o Elasticsearch, pudemos inserir um objeto mais completo, contendo os nomes e descrições dos pontos dos *templates*, resultando em uma busca mais precisa e abrangente. Já na implementação em Kotlin, a busca acessa títulos, descrições e títulos dos mapas dos *templates*, garantindo que os resultados sejam relevantes mesmo quando a consulta do usuário não se alinha perfeitamente com os termos armazenados.

É importante notar que, como o teste foi realizado com uma base de dados de teste e nenhum dado confidencial estava envolvido, utilizamos uma função de busca que ignorava a privacidade dos *templates*. Estes *templates* possuem um valor de privacidade que varia entre público e privado; porém, para fins de teste e avaliação da eficácia da busca, consideramos todas os *templates* indiscriminadamente.

Além disso, o problema principal da implementação atual em Kotlin é sua "ingenuidade", como visto na figura abaixo, ela realiza a busca em apenas um campo e não utiliza nenhum tipo de técnica que possa melhorar os resultados. Comparativamente, a nova lógica de busca separa as palavras para que elas sejam buscadas em ordens diferentes e em campos diferentes; tenta garantir que a *query* não foi escrita de maneira errada utilizando lógica fuzzy; observa se existe algum resultado próximo relevante. Essa relevância é determinada por diversos cálculos como *Field Weighting* , *Phrase Matching* e *Fuzzy Score* e, para que o resultado seja considerado satisfatório, é necessário que ele atinja um valor mínimo previamente estabelecido, tal limite é necessário para garantir uma boa precisão. Além disso, são implementadas técnicas como *term filtering* e *Regex Search* para reduzir ruído e garantir maior flexibilidade no momento de escrita da *query*.Desse modo mantemos a estrutura de busca sintática a aprimorando.



(a) Antiga lógica da busca de templates

(b) Nova lógica da busca de templates

4.4 Criação e implementação do *Tractatus*

Paralelamente ao desenvolvimento do mecanismo de busca tradicional, criamos um documento em formato `.txt` contendo todos os objetos *templates* magnificados, estes são os documentos que contém também os pontos de divergência nos mapas de cada *template*. Com esse documento completo pudemos utilizar um assistente implementado no ChatGPT que transforma esse arquivo em um *Tractatus* gradualmente. Nesse caso utilizamos o chat para gerar o *Tractatus* e ele conseguia gerar aproximadamente 1500 caracteres por requisição. A utilização de uma API com o mesmo prompt aceleraria o processo, pois ele consegue retornar 4000 tokens por requisição, o que é equivalente a 18.000 caracteres de acordo com o tokenizer da OpenAI (2024). Isso seria equivalente à metade do *Tractatus* final do documento original por requisição.

Este *tractatus* foi utilizado para treinar um assistente de suporte de busca utilizando modelos de linguagem de grande escala (LLM), nesse caso utilizamos novamente o ChatGPT4o pela sua

capacidade de receber arquivos, habilidade que o o1-mini e o1-preview não têm até o momento de publicação deste artigo.

Este Tractatus pode ser utilizado tanto de forma conectada quanto desconectada da API de *Strateegia*. Para empregá-lo diretamente, é necessário integrar a API de *Strateegia* à API do assistente e ajustar o prompt para que suas respostas sejam estruturadas de maneira a serem prontamente utilizadas para buscas dentro de *Strateegia*. Essa abordagem transforma a busca sintática em uma busca semântica de dois passos.

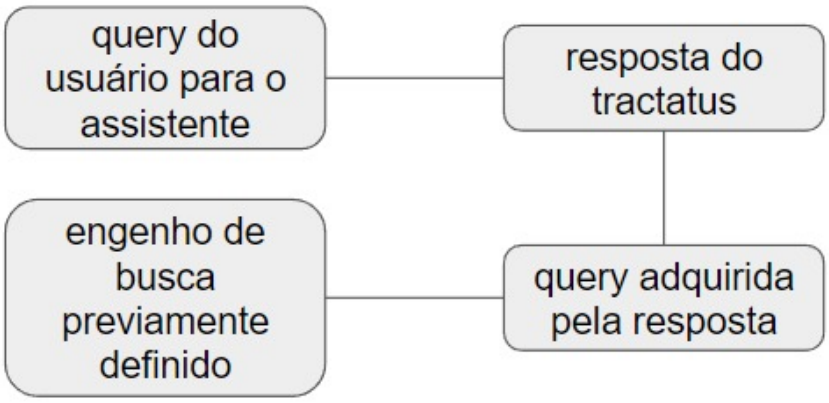


Figura 5: Exemplo da implementação de tractatus em conjunto com a busca sintática

5 Validação Do Sistema de Busca por Relevância

Os experimentos realizados para a avaliação do mecanismo de busca desenvolvido na plataforma *Strateegia* foram conduzidos em um ambiente computacional específico, visando garantir a consistência e a reprodutibilidade dos resultados. A seguir, detalharemos a configuração dos experimentos, as métricas de avaliação utilizadas, os resultados esperados, os resultados obtidos e a análise desses resultados.

5.1 Configuração do Experimento

Os experimentos foram conduzidos utilizando um computador Dell Inspiron equipado com um processador Intel(R) Core(TM) i7-1165G7 de 11ª geração, operando a 2.80 GHz (1.69 GHz em modo base), e 16GB de memória RAM. Essa configuração de hardware foi escolhida para proporcionar um ambiente de teste robusto, capaz de suportar as demandas computacionais inerentes às operações de busca e processamento de dados do mecanismo desenvolvido.

5.2 Métricas de Avaliação

A efetividade de um mecanismo de busca depende fundamentalmente da capacidade de fornecer resultados relevantes e completos ao usuário. Para medir essa efetividade, utilizam-se métricas como precisão, recall e F1 Score. Conforme discutido por Manning, Raghavan e Schütze em Introduction to Information Retrieval (Manning et al., 2008), a precisão (*Precision*) se refere

à proporção de documentos relevantes entre os resultados retornados pelo sistema. Ou seja, ela avalia o quão bem o sistema evita retornar resultados irrelevantes, sendo especialmente importante para garantir que os usuários recebam informação pertinente às suas consultas.

Por outro lado, o recall avalia a proporção de documentos relevantes recuperados em relação ao total de documentos relevantes disponíveis na base de dados. Essa métrica é essencial para garantir que o sistema não omita informação importante que poderia atender à necessidade do usuário (Manning et al., 2008).

A F1 Score, por sua vez, é a média harmônica entre precisão e recall, oferecendo uma medida balanceada da eficácia do sistema, especialmente em situações onde há necessidade de um equilíbrio entre os dois aspectos (Manning et al., 2008). Esta métrica é útil quando o sistema deve ser avaliado tanto pela qualidade dos resultados quanto pela abrangência de sua recuperação.

Essas métricas são fundamentais para compreender o desempenho de sistemas de busca e, consequentemente, para aprimorar a experiência do usuário, garantindo que a informação recuperada seja não apenas relevante, mas também abrangente.

5.3 Design dos experimentos

Os experimentos compartilham uma estrutura comum, utilizando *TF-IDF* (Hakim et al., 2014) para vetorização dos documentos e geração de queries de teste. O processo de criação do conjunto de testes foi similar nas três implementações: seleção aleatória de documentos da base, extração dos termos mais relevantes via *TF-IDF*, e estabelecimento de ground truth através de similaridade por cosseno ou categorias pré-existentes. Esta abordagem permitiu uma comparação justa entre as diferentes implementações, evidenciando que a lógica fundamental de busca permaneceu consistente independente da tecnologia utilizada.

5.4 Resultados Esperados

Nos experimentos conduzidos, espera-se que o mecanismo de busca apresente um desempenho relativamente baixo quando aplicado à base de dados real, devido à sua baixa qualidade e alta contaminação com informação de teste. Por outro lado, na base de dados utilizada exclusivamente para testes, com dados limpos e bem estruturados, espera-se um desempenho significativamente melhor, validando a eficácia das funções de busca implementadas tanto em Python quanto em Kotlin. Considerando as práticas modernas, valores de precisão na faixa de 0,70 a 0,90 e de recall entre 0,60 e 0,80 são considerados razoáveis e indicativos de um sistema eficiente (Manning et al., 2008).

5.5 Resultados dos Testes

Os experimentos realizados visam avaliar o desempenho das funções de busca implementadas em diferentes contextos e bases de dados. Como *benchmark* utilizamos os testes realizados na base de dados `fetch_20newsgroups` (Scikit-learn, 2024), que, por apresentar um nível reduzido de

ruído, tem resultados superiores aos realizados na base de dados de *templates*. Os testes foram executados múltiplas vezes para garantir a estabilidade dos resultados finais.

Na função de busca aplicada a essa base, a precisão variou entre 0,91 e 1, o *recall* entre 0,82 e 1, e a medida F1 também entre 0,82 e 1. Em diversos testes, os resultados atingiram o valor máximo de 1; contudo, essa replicabilidade completa depende da seleção das consultas de teste e não é consistentemente alcançada. Esses resultados são considerados excelentes e demonstram que a lógica da função de busca possui um sólido embasamento, funcionando com alta qualidade quando a base de dados utilizada possui um nível de organização adequado.

Além disso, realizamos os testes com o *Tractatus* gerado com o Chat GPT 4o. Os resultados foram satisfatórios e as recomendações são alinhadas com o que um usuário com conhecimento escolheria utilizar entre as possibilidades disponíveis, as justificativas fazem sentido e explicam bem as decisões tomadas, tais justificativas estão alinhadas com o que foi implementado no *template* recomendado. Devido ao tamanho das respostas os exemplos estão disponibilizados no fim do texto na seção de apêndices.

Tabela 1: Resultados dos testes com python com a base de fetch_20newsgroups

título	**Número de Queries**	**Recall**	**Precision**	**F1-Score**
teste 1	100	1,00	1,00	1,00
teste 2	100	0,88	0,96	0,88
teste 3	100	0,88	0,93	0,88

Em seguida, avaliamos a função de busca em Python utilizando o Elasticsearch na base de dados de *templates* de *Strateegia*. Os resultados obtidos foram consideravelmente inferiores em comparação com os da base `fetch_20newsgroups`, apresentando maior variabilidade devido ao ruído presente nos dados. A precisão variou entre 0,20 e 0,36, o *recall* entre 0,28 e 0,71, e a medida F1 entre 0,22 e 0,48. Esses resultados dependem significativamente das consultas selecionadas, pois alguns objetos estão mais sujeitos ao ruído do que outros. Apesar disso, considerando a quantidade de objetos repetidos e inconsistentes que prejudicam os resultados, esses valores são razoáveis e não refletem necessariamente a real qualidade do código.

É importante destacar que, ao realizar buscas com termos coerentes, como pesquisar o *template* "Bluu Ocean" pelo termo "Blue Ocean", o sistema retorna apenas o *template* correto, atingindo o objetivo principal do mecanismo de busca. Em testes realizados com um maior número de consultas, os resultados tendem a se estabilizar em torno de 0,40 de precisão, 0,58 de *recall* e 0,43 de medida F1.

Tabela 2: Resultados dos testes com python na base de dados de templates

título	**Número de Queries**	**Recall**	**Precision**	**F1-Score**
teste positivo	10	0,71	0,36	0,48
teste negativo	10	0,49	0,29	0,35
teste normalizado	50	0,55	0,37	0,41

O último conjunto de testes foi conduzido na implementação em Kotlin, executando os serviços localmente, o que resultou em uma velocidade significativamente maior, especialmente

em comparação com os testes anteriores, devido à possibilidade de realizar um maior número de consultas em menos tempo. Os resultados apresentaram maior variabilidade, com a precisão variando entre 0,08 e 0,46, o *recall* entre 0,17 e 0,57, e a medida F1 entre 0,11 e 0,50. Novamente, ao ampliar a quantidade de consultas nos testes, os resultados tendem a se estabilizar em valores entre os extremos. Nesse caso eles são próximos a 0,35 de precisão, 0,51 de *recall* e 0,40 de medida F1.

Tabela 3: Resultados dos testes com kotlin no na base de dados de templates

título	**Número de Queries**	**Recall**	**Precision**	**F1-Score**
teste positivo	10	0,57	0,46	0,50
teste negativo	10	0,17	0,08	0,11
teste normalizado	50	0,54	0,37	0,42

De modo geral, os resultados indicam que a utilização de técnicas do Elasticsearch e a implementação de algoritmos em Kotlin para simular essas funções produzem desempenhos extremamente similares. Para a funcionalidade proposta, não se faz necessária a aplicação de técnicas complexas, especialmente considerando o caráter de prova de conceito desta pesquisa. A similaridade dos resultados na base de dados de teste sugere que desempenhos equivalentes podem ser esperados quando implementados em um ambiente com acesso a uma base de dados de maior qualidade, como a de produção de *Strateegia*.

Esses resultados corroboram a eficácia das abordagens adotadas e reforçam a importância de dispor de bases de dados organizadas e com baixo nível de ruído para otimizar o desempenho dos mecanismos de busca. A implementação futura em um ambiente de produção, com dados de maior qualidade, tende a melhorar os índices de precisão e *recall*, potencializando a usabilidade e a satisfação dos usuários ao interagir com a plataforma.

6 Conclusão

Comparando os resultados dos testes das implementações em Kotlin e Python, observamos que ambos os mecanismos de busca apresentaram desempenho extremamente similar. Isso era esperado, pois as funções desenvolvidas são essencialmente espelhadas uma da outra, adaptadas apenas para diferentes linguagens de programação. Essa consistência nos resultados indica que, quando o código em Kotlin for implementado em produção, é provável que apresente qualidade comparável à solução baseada em Elasticsearch, especialmente ao lidar com requisições mais simples e direta, é possível afirmar isso devido ao resultado muito bom da implementação de Elasticsearch com a base de dados NewsGroups.

Em relação ao *Tractatus*, os testes demonstraram uma boa capacidade de raciocínio, respondendo de maneira rápida e coerente e fornecendo explicações claras e detalhadas, tanto as respostas quanto às explicações estavam alinhadas com as respostas esperadas para os prompts. Além disso, é possível ajustar o *prompt* utilizado para que o assistente retorne apenas o título do *template* desejado, facilitando a busca subsequente. Mesmo quando são feitas perguntas

extremamente ingênuas ou vagas, o *Tractatus* é capaz de fornecer respostas relevantes e elucidativas, evidenciando sua potencial eficácia em auxiliar usuários leigos na navegação pela plataforma.

Os resultados obtidos reforçam a eficácia das técnicas de busca implementadas e o potencial de incorporar mais assistentes baseados em LLMs para melhorar a experiência do usuário. A adoção dessas soluções pode contribuir para a usabilidade da plataforma *Strateegia*, atendendo aos objetivos propostos de aprimorar a exploração da plataforma, de reduzir a barreira de conhecimento para novos usuários e de atualizar a função de busca sintática antiquada.

6.1 Limitações e Trabalhos Futuros

Apesar dos resultados positivos, a implementação do Elasticsearch apresenta limitações, principalmente devido à necessidade de gerenciar uma infraestrutura adicional. A manutenção de um índice atualizado requer o controle constante de mais uma base de dados e a dependência de um sistema extra em funcionamento. Para mitigar esse desafio, consideramos a utilização do Atlas Search, uma ferramenta integrada ao MongoDB que oferece funcionalidades avançadas de busca (MongoDB, 2024).

Embora os testes com o Atlas Search não tenham sido possíveis neste projeto, devido à sua disponibilidade apenas na nuvem, planeja-se sua implementação futura. O Atlas Search permite uma melhor implementação de técnicas como busca aproximada (*fuzzy search*), autocompletar, recomendação de documentos similares e gerenciamento avançado de sinônimos. Além disso, suporta o uso de índices, o que não é viável com o MongoDB convencional ao utilizar busca aproximada. Isso simplifica o processo e melhora os resultados das buscas, otimizando o desempenho do sistema.

Um problema mais significativo reside na estrutura atual dos objetos do projeto, que não é otimizada para operações de busca. Para realizar uma busca similar à implementada no Elasticsearch, seria necessário realizar múltiplas consultas separadas e agregá-las, resultando em um uso ineficiente de recursos computacionais. Portanto, é necessário refatorar a estrutura dos objetos, centralizando toda informação importante, a fim de melhorar a eficiência e a escalabilidade das buscas.

No que concerne ao *Tractatus*, é fundamental estabelecer uma conexão direta com o mecanismo de busca e implementar um processo de atualização automática, garantindo que ele reflita as mudançaFs e adições na base de dados. Na plataforma *Strateegia*, já existe a utilização de assistentes para a criação de respostas e pontos. Ao utilizar *prompts* específicas, é possível ajustar as respostas do assistente para retornar apenas a informação necessária para a busca, como o título do *template*.

Para manter o *Tractatus* atualizado, propomos o uso de uma *prompt* especializada na geração automática do documento, aliada a um sistema de verificação periódica que detecta atualizações significativas na base de dados. Esse processo é similar à atualização dos índices no Elasticsearch, mas acreditamos que a manutenção do *Tractatus* traz um valor agregado maior, dado seu impacto positivo na usabilidade e na experiência do usuário.

Como trabalhos futuros, planejamos implementar assistentes similares e aprimorar os mecan-

ismos de busca em outros elementos da plataforma que sejam relevantes, como a busca por jornadas ou por questões dentro de uma jornada. Acreditamos que essas melhorias contribuirão para tornar *Strateegia* uma plataforma ainda mais intuitiva e acessível, alinhada com as necessidades dos usuários e as tendências atuais em interação humano-computador.

6.2 Uso de Assistentes de Inteligência Artificial no Desenvolvimento de Código

Tendo em vista que este artigo apoia, cria soluções utilizando e justifica o uso de LLM's para desenvolvimento, aprendizado e apoio, uma parte significativa do desenvolvimento deste projeto envolveu o uso de assistentes de inteligência artificial, especificamente o *Claude 3.5 Sonnet*, para a geração e modificação de código. O *Claude* foi extremamente útil na criação de código básico de alta qualidade, que pôde ser testado e modificado de forma iterativa. A ferramenta disponibiliza *artifacts* que oferecem suporte substancial para o desenvolvimento ágil, permitindo gerar mais de 30 versões para cada função e transformá-las em funções diferentes rapidamente.

Essa capacidade de modificar massivamente o código acelerou significativamente o projeto. A necessidade de testar em outra base de dados, percebida no meio do desenvolvimento, foi atendida com facilidade graças ao *Claude*. Bastava apresentar o código existente e solicitar a modificação da base de dados, mantendo a lógica de busca. Embora o *Claude* possa cometer deslizes, a intervenção humana foi essencial para revisar e ajustar o código gerado, garantindo a qualidade e a funcionalidade desejadas.

Além disso, o *Claude* demonstrou ser perceptivo quanto às possíveis necessidades e problemas que poderiam ser encontrados ao utilizar determinadas implementações. Por exemplo, ao considerar a utilização do Atlas Search, o *Claude* informou prontamente sobre a inviabilidade dessa abordagem no contexto local, uma vez que o Atlas Search funciona apenas em nuvem. Essa orientação foi crucial, pois evitou investimentos desnecessários de tempo em uma solução que não atenderia aos requisitos do projeto, que visava desenvolver um *proof of concept* sem depender de serviços em nuvem. Ao testar as mesmas requisições com o *ChatGPT o1-preview*, ele recomendava funções às vezes até melhores que o *Claude*, mas não parecia ter a mesma capacidade do *Claude* de avisar o usuário das possíveis adversidades.

Inicialmente, o projeto também contemplava a implementação de uma *pipeline* de dados para atualizar constantemente o Elasticsearch com novos *templates* inseridos no MongoDB. Após enfrentar diversos problemas nessa tentativa, devido principalmente às funcionalidades serem desenvolvidas com Linux em mente ao invés de Windows, a utilização de *Claude* contribuiu para a resolução desses obstáculos técnicos. No entanto, em conjunto com o orientador, decidimos que, ao compreender que este projeto é uma prova de conceito, não seria necessário desenvolver uma funcionalidade destinada apenas à atualização contínua da base de dados.

Apêndices

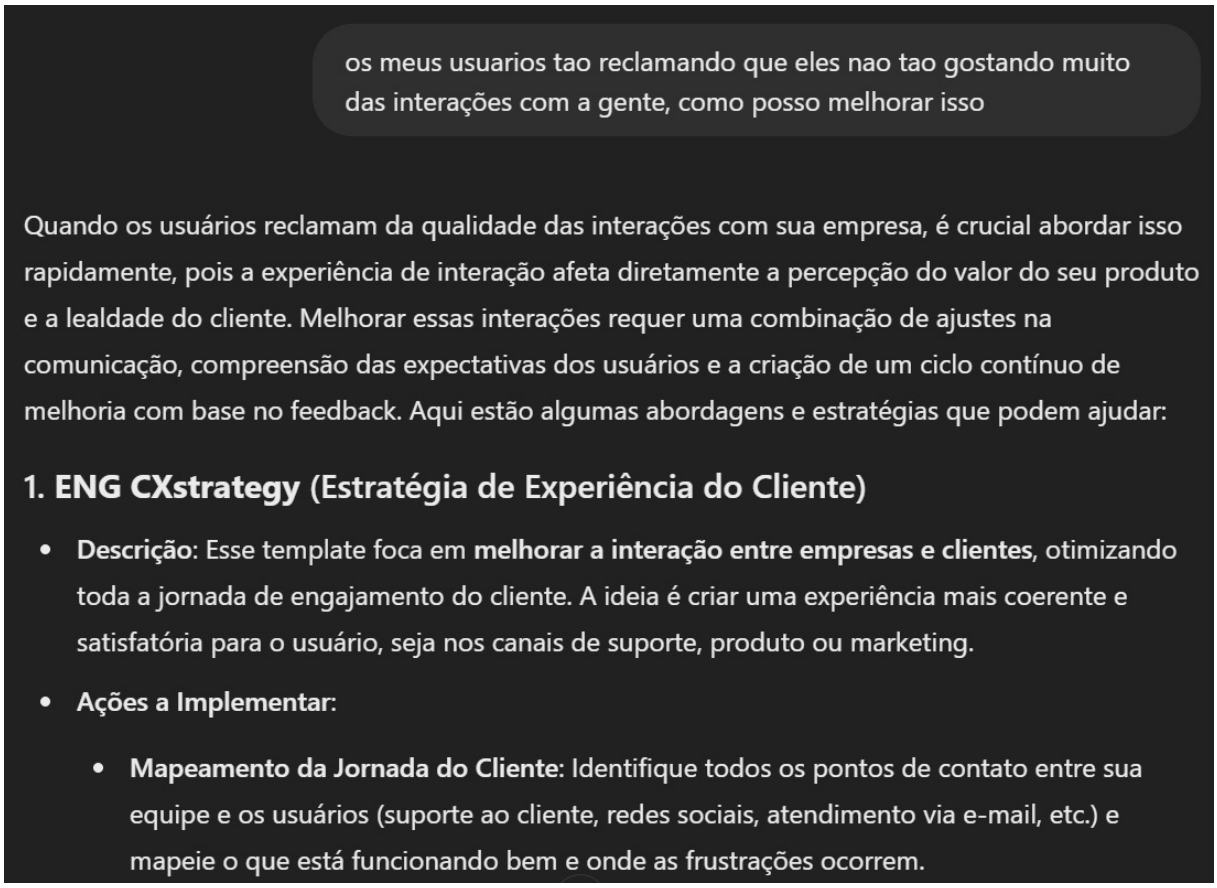


Figura 6: Exemplo de requisição no Chat GPT utilizando o Tractatus

Na figura 6, o Usuário explicita a necessidade urgente de aprimorar a comunicação com os usuários do seu produto, nessa situação o assistente contextualizado compreende e retorna uma solução baseada na informação baseada no *Tractatus*, nesse caso recomendando o *Template CXstrategy* em inglês(ENG), essa sendo um *template* que implementa um plano organizacional para gerenciar e aprimorar interações e percepções dos clientes. Essa resposta foi considerada adequada por também explicar os passos necessários.

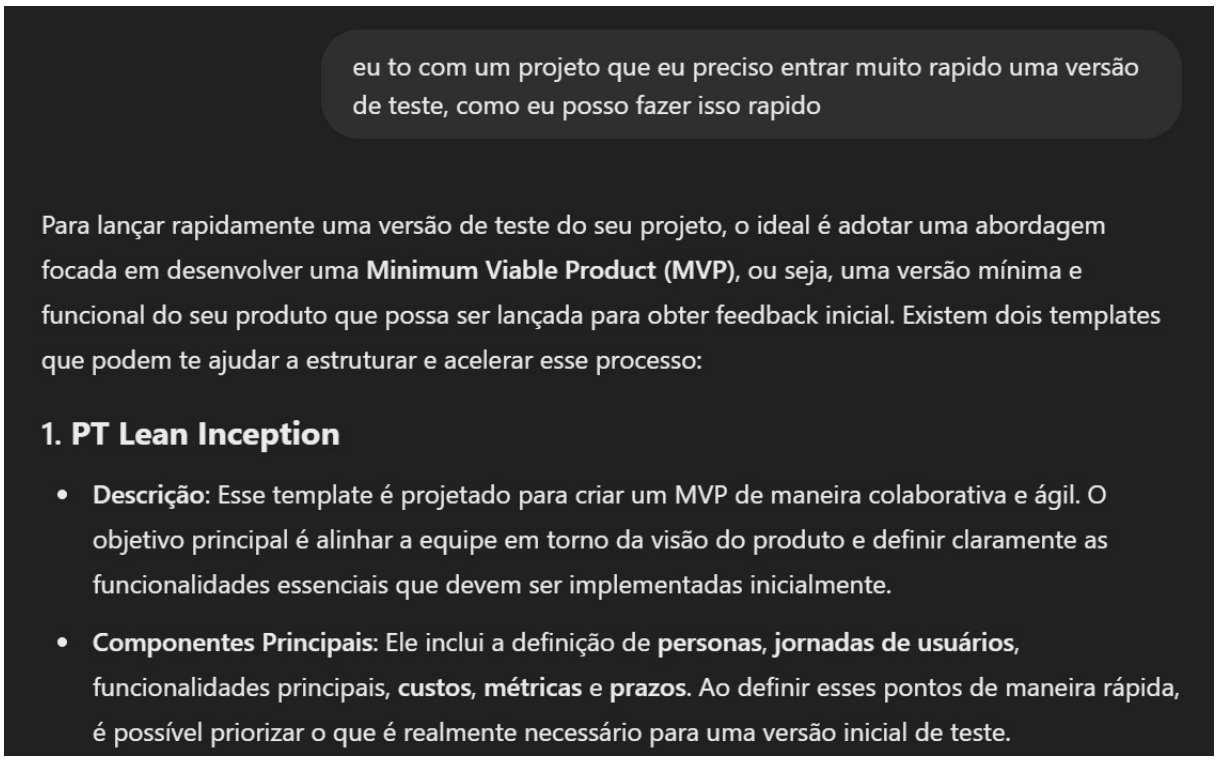


Figura 7: Exemplo de requisição no Chat GPT utilizando o Tractatus

Na figura 7 foi requisitado um *Template* que ajudasse o usuário a criar uma versão de texte de um projeto, compreendendo que uma versão rápida de teste é um MVP o Assistente recomendou a atualização de *PT Lean Inception*, essa técnica tenta criar uma solução de maneira colaborativa e ágil. Essa resposta foi considerada adequada por também descrever o que seria realizado ao longo desse processo.

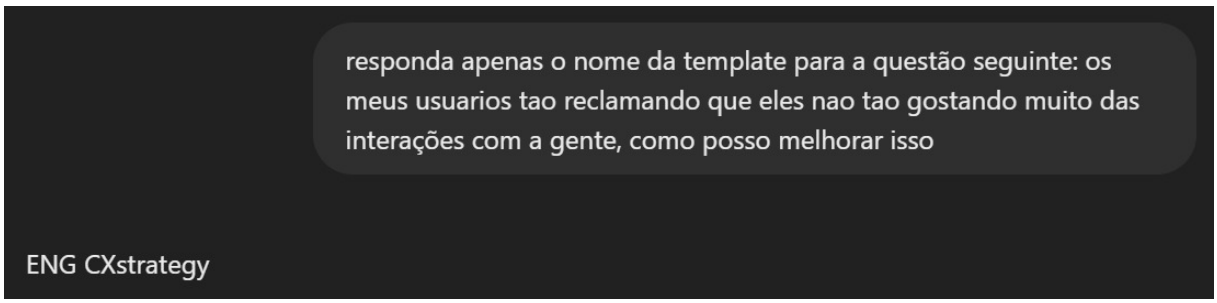


Figura 8: Exemplo de requisição no Chat GPT utilizando o Tractatus com a necessidade de retornar apenas o título do template requisitado

Na figura 8 o teste foi fazer uma requisição similar a figura 4 mas dessa vez a requisição conteria uma necessidade extra, sendo ela o pedido para o assistente comunicar apenas o nome do *Template* recomendado. Essa resposta foi considerada adequada por responder apenas o nome do *Template* da figura 4.

References

Baeza-Yates, R., & Ribeiro-Neto, B. (1999). *Modern information retrieval*. Boston, MA: Addison-Wesley.

Banathy, B. H. (1996). *Designing social systems in a changing world*. Boston, MA: Springer. Retrieved from https://link.springer.com/book/10.1007/978-1-4757-9981-1 (Accessed: 2024-10-13) DOI: 10.1007/978-1-4757-9981-1

Design Council. (2007). *Eleven lessons: Managing design in eleven global brands*. Retrieved from https://www.designcouncil.org.uk/fileadmin/uploads/dc/Documents/ElevenLessons_Design_Council%2520%25282%2529.pdf (Accessed: 2024-10-13)

Dong, Y., Jiang, X., Jin, Z., & Li, G. (2024, September). Self-collaboration code generation via chatgpt. *ACM Transactions on Software Engineering and Methodology*, *33*(7), 1–38. Retrieved from https://doi.org/10.1145/3672459 DOI: 10.1145/3672459

Fasolo, B., McClelland, G. H., & Todd, P. M. (2007). Escaping the tyranny of choice: when fewer attributes make choice easier. *Marketing Theory*, *7*(1), –. DOI: 10.1177/1470593107073842

Fernandes, M. R. (2001). *Uma abordagem visando escalabilidade para crawling e indexação* (Master's thesis, Universidade Federal de Pernambuco, Recife). Retrieved from https://repositorio.ufpe.br/bitstream/123456789/2545/1/arquivo4931_1.pdf (Accessed: 2023-11-21)

Garcia, V. C., Lucrédio, D., Durão, F. A., Santos, E. C. R., Almeida, E. S. d., Fortes, R. P. d. M., & Meira, S. R. d. L. (2006). From specification to experimentation: A software component search engine architecture. In *Lecture notes in computer science* (pp. 82–97). Springer. Retrieved from https://www.cin.ufpe.br/~alt/mestrado/papers/From_the_specification.pdf DOI: $10.1007/11783565_6$

Gigerenzer, G., & Goldstein, D. G. (1996). Reasoning the fast and frugal way: models of bounded rationality. *Psychological Review*.

Goldstein, D. G., & Gigerenzer, G. (2002). Models of ecological rationality: the recognition heuristic. *Psychological Review*, *109*(1), 75–90. DOI: 10.1037/0033-295x.109.1.75

Hakim, A. A., Erwin, A., Muliady, W., & outros. (2014, Outubro). Automated document classification for news articles in bahasa indonesia based on term frequency inverse document frequency (tf-idf) approach. In *International conferences on [nome completo da conferência]*. IEEE. DOI: 10.1109/ICITEED.2014.7007894

Lee, D., Moon, J., & Kim, Y. (2007). The effect of simplicity and perceived control on perceived ease of use. In *Amcis 2007 proceedings*. Retrieved from https://aisel.aisnet.org/amcis2007/71

Linden, G., Smith, B., & York, J. (2003). Amazon.com recommendations: item-to-item collaborative filtering. *IEEE Internet Computing*, *7*(1), 76–80.

Liu, Z., Zou, L., Zou, X., Wang, C., Zhang, B., Tang, D., ... Cheng, Y. (2022). *Monolith: Real time recommendation system with collisionless embedding table.* Bytedance Inc.

Manning, C. D., Raghavan, P., & Schütze, H. (2008). *Introduction to information retrieval.* Cambridge, UK: Cambridge University Press.

Meira, S. (2021). *Nem real, nem virtual: o mundo é figital.* Retrieved from https://silvio.meira.com/nem-real-nem-virtual-o-mundo-e-figital/ (Accessed: 2024-10-13)

MongoDB. (2024). Atlas search [Computer software manual]. Retrieved from https://www.mongodb.com/atlas/search (Accessed: 11/10/2024)

Naveed, H., Khan, A. U., Qiu, S., Saqib, M., Anwar, S., Usman, M., ... Mian, A. (2023). A comprehensive overview of large language models. *arXiv preprint arXiv:2307.06435*. Retrieved from https://arxiv.org/abs/2307.06435

Neves, A., Meira, S., Calegário, F., Meira, P., & Brito, A. (2024). *Tractatus: A wittgenstein-inspired framework for structuring knowledge in llms.* Retrieved from https://www.tds.company/blog/tractatus-a-wittgenstein-inspired-framework-for-structuring-knowledge-in-llms (Available at TDS Company Blog)

Neves, A., Meira, S., Medeiros, L., Ferraz, M., Soter, C., Cavalcanti, S., ... Heimann, V. (2020). Strateegia.digital: A platform that assumes design as a strategic tool. In *Lecture notes in computer science* (Vol. 12200). DOI: 10.1007/978-3-030-49713-$2_1$5

OpenAI. (2024). *Openai tokenizer.* https://platform.openai.com/tokenizer. (Accessed: 10/10/2024)

Osterwalder, A. (2008). *The business model canvas.* Retrieved from https://nonlinearthinking.typepad.com/nonlinear_thinking/2008/07/the-business-model-canvas.html (Accessed: 15 October 2024)

Page, L., & Brin, S. (1998). The anatomy of a large-scale hypertextual web search engine. In *Seventh international world-wide web conference (www 1998).*

Page, L., Brin, S., Motwani, R., & Winograd, T. (1999). The pagerank citation ranking: Bringing order to the web. *Technical Report*, *1999-66*. Retrieved from http://ilpubs.stanford.edu:8090/422/

Peng, B., Zhu, Y., Liu, Y., Bo, X., Shi, H., Hong, C., ... Tang, S. (2024). Graph retrieval-augmented generation: A survey. *arXiv preprint arXiv:2408.08921*. Retrieved from https://doi.org/10.48550/arXiv.2408.08921

Scikit-learn. (2024). *sklearn.datasets.fetch_20newsgroups.* https://scikit-learn.org/1.5/modules/generated/sklearn.datasets.fetch_20newsgroups.html. (Acesso em: 13 out. 2024)

tds.company. (2024). *Strateegia digital platform.* https://strateegia.digital/. (Ac-

cessed: 10-10-2024)

Wittgenstein, L. (1922). *Tractatus logico-philosophicus* (C. K. Ogden, Trans.). London: Routledge & Kegan Paul.

Zadeh, L. A. (1965). Fuzzy sets. *Information and Control*, *8*(3), 338–353.

Zamfir, V.-A., Carabas, M., Carabas, C., & Tapus, N. (2019). Systems monitoring and big data analysis using the elasticsearch system. *Proceedings of the 22nd International Conference on Control Systems and Computer Science*, 481–486.