Arthur Andrade e Davi Henrique

Avaliação de métodos estatísticos na busca de correspondência textual jurídica frente a *embeddings*

São Paulo - Brasil

2025

Arthur Andrade e Davi Henrique

Avaliação de métodos estatísticos na busca de correspondência textual jurídica frente a *embeddings*

Monografia apresentada na disciplina Trabalho de Conclusão de Curso, como parte dos requisitos para obtenção do título de Bacharel em Ciência da Computação.

Centro Universitário Senac - Santo Amaro
Bacharelado em Ciência da Computação

Orientador: Nome do Orientador

São Paulo - Brasil
2025

"A beleza é realmente um bom dom de Deus; mas que os bons não pensem que ela é um grande bem, pois Deus a distribui mesmo para os maus.
(Santo Agostinho)

Lista de ilustrações

Lista de abreviaturas e siglas

API	Application Programming Interface
BERT	Bidirectional Encoder Representations from Transformers
BMSS	Boyer-Moore String Search
KMP	Knuth Morris Pratt
CBOW	Continuous Bag-of-Words
CNJ	Conselho Nacional de Justiça
CPU	Computing Processing Unit
GPU	Graphics Processing Unit
CRF	Conditional Random Field
ELMo	Embeddings from Language Model
GO	Golang Programming Language
GloVe	Global Vectors for Word Representation
HTTP	Hypertext Transfer Protocol
IDF	Inverse Document Frequency
JSON	JavaScript Object Notation
NLP	Natural Language Processing
NER	Reconhecimento de Entidade Nomeada
POS	Part-of-Speech
PJe	Processo Judicial Eletrônico
SVM	Support Vector Machine
RAG	Retrieval-Augmented Generation
REST	Representational State Transfer
TF-IDF	Term Frequency-Inverse Document Frequency

Sumário

1 Introdução

1.1 Contexto

A era digital transformou radicalmente o acesso à informação, gerando um volume de dados sem precedentes em praticamente todos os setores da sociedade. No campo do Direito, essa realidade se manifesta na digitalização massiva de processos, leis, jurisprudências e pareceres. O Judiciário brasileiro, por exemplo, recebeu mais de 250 milhões de processos em formato eletrônico nos últimos 15 anos (Agência CNJ de Notícias, 2024a). Se por um lado essa digitalização representa um avanço em termos de transparência, por outro, impõe um desafio monumental: como navegar e extrair informações relevantes de um oceano de textos complexos e interconectados? A simples capacidade de armazenamento tornou-se insuficiente; a necessidade agora é de processamento inteligente.

Nesse cenário, as técnicas de Processamento de Linguagem Natural (NLP), especialmente aquelas baseadas em Inteligência Artificial e aprendizado de máquina, surgiram como a solução mais promissora. Modelos de linguagem avançados, capazes de compreender nuances semânticas e contextuais, prometem revolucionar a busca jurídica. Ferramentas que utilizam *embeddings* e arquiteturas de *transformers* representam o estado da arte, buscando emular uma compreensão quase humana dos textos legais para tarefas como a identificação de divergências jurisprudenciais (PAULA et al., 2024).

Contudo, a adoção dessas tecnologias levanta uma questão fundamental e muitas vezes negligenciada: qual o custo-benefício real dessa complexidade? O esforço computacional, financeiro e técnico para treinar, implementar e manter sistemas de IA é significativo, exigindo a aplicação de métodos para otimizar e reduzir os custos de inferência dos modelos (DATACAMP, 2025). Isso nos leva ao cerne deste trabalho: investigar até que ponto a sofisticação dos modelos de *machine learning* é indispensável para a tarefa de busca em documentos jurídicos. Será que métodos estatísticos e algoritmos de busca mais clássicos, quando bem aplicados, não poderiam alcançar resultados comparáveis, com uma fração do custo?

Este estudo se propõe a explorar essa fronteira. Conforme aponta Santos (2023), modelos distribucionais como BERT não apresentam, necessariamente, um desempenho superior em todas as tarefas de classificação de textos jurídicos quando comparados a abordagens mais simples. O objetivo deste trabalho, portanto, não é desenvolver uma nova ferramenta, mas sim realizar uma análise crítica e comparativa, buscando compreender o verdadeiro *trade-off* entre a precisão semântica e a eficiência computacional no domínio específico do Direito.

1.2 Justificativa

A busca incessante por precisão levou o campo da tecnologia jurídica a uma crescente dependência de modelos de Inteligência Artificial, cujo uso no Judiciário brasileiro cresceu 26% apenas entre 2022 e 2023 (Agência CNJ de Notícias, 2024b). A premissa é clara: quanto mais "inteligente"o modelo, melhores os resultados. No entanto, essa corrida pela sofisticação muitas vezes ignora os custos práticos associados. O treinamento de grandes

modelos de linguagem exige poder de processamento massivo, geralmente dependente de hardware especializado como GPUs, o que se traduz em altos custos de infraestrutura, energia e manutenção. Para muitas instituições, essa barreira pode ser intransponível.

Paralelamente, a evolução do hardware, com o advento das arquiteturas *multicore*, não beneficiou apenas os modelos de IA. Algoritmos clássicos, antes limitados por execuções sequenciais, podem hoje ter seu desempenho drasticamente otimizado por meio de paralelismo, explorando múltiplos núcleos de processamento para aumentar a performance das aplicações (FLECK, 2012). Técnicas estatísticas como o TF-IDF e algoritmos de busca de padrões, como o Boyer-Moore, são computacionalmente mais leves e mais simples de implementar, mas seu potencial em hardware moderno é frequentemente subestimado.

A justificativa deste trabalho reside, portanto, em uma necessidade de pragmatismo. É preciso questionar se o ganho marginal de relevância oferecido por um modelo semântico complexo justifica sua implementação em todos os cenários. Em domínios com linguagem controlada e previsível, como é o caso dos documentos jurídicos, abordagens mais simples, baseadas em regras, podem se mostrar altamente eficazes (SAP, s.d.). Em muitas tarefas cotidianas, a velocidade de resposta e o baixo custo operacional podem ser mais valiosos do que a capacidade de interpretar uma ambiguidade sutil em um texto.

Esta pesquisa oferece uma análise quantitativa desse *trade-off*. Ao comparar diretamente o desempenho, o tempo de resposta e o uso de recursos de diferentes abordagens — uma prática já adotada em trabalhos que avaliam modelos como BERT e ChatGPT por meio de métricas de acurácia, precisão e F1-score (NUNES; MENDONçA; RALHA, 2024) — pretendemos fornecer dados concretos que auxiliem na tomada de decisão sobre qual tecnologia é mais adequada para diferentes necessidades do ecossistema jurídico, movendo o debate do campo teórico para uma avaliação de aplicabilidade prática.

1.3 Objetivos

1.3.1 Objetivo Geral

Avaliar e comparar o desempenho de métodos estatísticos e semânticos para a indexação e busca de informações em documentos jurídicos, analisando o *trade-off* entre a complexidade computacional dos modelos e a relevância dos resultados obtidos.

1.3.2 Objetivos Específicos

1. Pré-processar um corpus de documentos jurídicos, incluindo limpeza textual e outras técnicas pertinentes, para criar uma base de testes consistente para ambas as abordagens.

2. Implementar um pipeline de busca baseado em *embeddings* de sentenças, utilizando a similaridade de cosseno para ranquear os resultados de acordo com a proximidade semântica.

3. Implementar um pipeline de busca baseado no modelo estatístico TF-IDF para criar uma representação vetorial baseada na frequência de termos.

4. Utilizar o algoritmo de busca de padrões Boyer-Moore para realizar buscas por correspondência exata de termos, servindo como uma linha de base de desempenho.

5. Definir e aplicar um conjunto de métricas para avaliar e comparar o desempenho das diferentes abordagens, considerando precisão, tempo de resposta e uso de recursos computacionais.

6. Analisar criticamente os resultados obtidos para identificar as vantagens, desvantagens e os cenários de aplicação mais adequados para cada método no contexto específico dos textos jurídicos.

2 Revisão bibliográfica

2.1 Processamento de Linguagem Natural no Domínio Jurídico

A crescente produção textual na sociedade contemporânea transformou a informação em um recurso abundante, porém desafiador de ser processado. No universo jurídico, esse fenômeno é ainda mais acentuado: contratos, decisões, pareceres, petições e legislações são produtos essencialmente linguísticos. O Direito, enquanto sistema normativo, é construído, operado e interpretado por meio da linguagem (ABREU, 2022; MELLO, 2020).

Diante do volume exponencial de documentos jurídicos, o uso de ferramentas computacionais tornou-se uma necessidade. As limitações cognitivas humanas frente à grande quantidade de dados disponíveis exigem soluções capazes de automatizar parte do processamento textual. Técnicas de NLP surgem como abordagem consolidada para enfrentar esse desafio (ZAMPIERI et al., 2021; SOUZA, 2023).

O NLP, subárea da inteligência artificial, permite que máquinas compreendam, processem e operem sobre dados em linguagem natural. No contexto jurídico, essas técnicas viabilizam a identificação de padrões, a extração de informações relevantes, a sumarização de documentos e o suporte à tomada de decisão. Trata-se de uma evolução que não apenas automatiza tarefas, mas incorpora mecanismos capazes de operar diretamente sobre a base textual do Direito (PINHEIRO, 2023; MELLO, 2020).

Atualmente, modelos computacionais já demonstram aplicações práticas no ambiente jurídico, como análise de jurisprudência, redação assistida e busca semântica em bases normativas. No contexto brasileiro, estudos como os de Zampieri et al. (2021) evidenciam a consolidação dessas tecnologias no setor jurídico.

2.1.1 Características dos textos jurídicos

A aplicação de técnicas de NLP no domínio jurídico envolve desafios particulares, uma vez que os textos jurídicos apresentam propriedades distintas em relação a outros gêneros textuais. Em primeiro lugar, são documentos altamente especializados, redigidos com vocabulário técnico e linguagem formal, frequentemente estruturados de modo padronizado. Termos como *vossa excelência*, *ex positis* ou *ad nutum* não apenas indicam jargão, mas também desempenham papel semântico e pragmático central na construção do sentido legal de um texto (ABREU, 2022; SOUZA, 2023).

Além disso, esses textos apresentam um alto grau de ambiguidade controlada — ou seja, ambiguidade tolerada e, muitas vezes, intencional. Palavras como *poderá*, *deverá* ou *caberá* têm significados normativos específicos que impactam diretamente a interpretação e aplicação das normas. Essa polissemia exige que modelos de NLP sejam sensíveis ao contexto jurídico para evitar erros semânticos que possam comprometer a análise automatizada (ZAMPIERI et al., 2021; PINHEIRO, 2023).

Outro aspecto relevante é a intertextualidade jurídica. Leis, contratos, petições e sentenças costumam referenciar outros documentos legais, criando dependências explícitas entre normas, jurisprudências e atos administrativos. Essa característica impõe a

necessidade de modelos capazes de capturar relações entre documentos — como citações, remissões e precedentes — o que exige abordagens sofisticadas para modelagem de contexto e representação semântica (ZAMPIERI et al., 2021).

Também é importante considerar a estrutura argumentativa dos textos jurídicos. Documentos como sentenças e pareceres possuem uma lógica discursiva específica, geralmente organizada em premissas, fundamentações e conclusões. Essa organização influencia tanto a segmentação quanto a priorização de informações nos sistemas de NLP aplicados ao Direito (MELLO, 2020).

2.1.2 Desafios específicos em português

Ao aplicar técnicas de NLP à língua portuguesa, especialmente no domínio jurídico, surgem obstáculos adicionais que não estão presentes em línguas com estruturas morfossintáticas mais simples, como o inglês. O português brasileiro apresenta uma morfologia altamente flexiva, com ampla variação de formas verbais, nominais e pronominais, o que torna tarefas como tokenização e lematização consideravelmente mais complexas (ZAMPIERI et al., 2021).

No contexto jurídico, essa complexidade é intensificada pelo uso recorrente de locuções e expressões idiomáticas específicas da tradição normativa. Frases como *sem prejuízo do disposto*, *salvo disposição em contrário* ou *nos termos da legislação aplicável* não podem ser analisadas isoladamente por palavra. Tais construções funcionam como unidades semânticas coesas, exigindo abordagens capazes de identificar e preservar o significado completo durante o pré-processamento textual (SOUZA, 2023; ZAMPIERI et al., 2021).

2.1.3 Iniciativas e ferramentas existentes

Uma das iniciativas voltadas ao português jurídico é o projeto *LegalNLP* (ZAMPIERI et al., 2021), que disponibiliza um conjunto de modelos pré-treinados — como Word2Vec, Doc2Vec, FastText e BERT — ajustados ao domínio legal brasileiro. O projeto inclui ainda um pacote em Python com funcionalidades específicas para tarefas como vetorização, extração de entidades e classificação de textos legais. Os modelos foram treinados com dados provenientes de diversos tribunais nacionais, o que visa proporcionar maior aderência linguística e semântica ao contexto jurídico do país, em comparação com abordagens mais genéricas.

Outras iniciativas relevantes incluem o *OpenLegalData*, o *JusBrasil Labs* e os repositórios públicos mantidos pelo *Conselho Nacional de Justiça* (CNJ) e pelos tribunais superiores, que fornecem bases de dados utilizadas em aplicações experimentais de NLP jurídico. No campo das ferramentas gerais de NLP, bibliotecas como *spaCy*, *NLTK*, *Stanza* e modelos da *Hugging Face* oferecem suporte para o português, embora possam demandar adaptações específicas ao domínio jurídico devido à complexidade e à terminologia especializada dos textos legais — um aspecto abordado por iniciativas como o *LegalNLP* (ZAMPIERI et al., 2021).

2.2 Técnicas de Pré-processamento Textual

Em um cenário de grande volume textual, como o jurídico, a extração de informação relevante demanda um tratamento prévio sobre os documentos. Essa etapa é conhecida como pré-processamento, e tem como objetivo transformar textos brutos em representações mais estruturadas e significativas, facilitando tanto a análise semântica quanto a indexação e recuperação posterior. Trata-se, portanto, de uma fase indispensável para qualquer sistema de recuperação de informação textual, sobretudo em domínios especializados, já que a escolha adequada das técnicas de pré-processamento pode influenciar significativamente a relevância das informações extraídas (OLIVEIRA; RODRIGUES; APPEL, 2022).

Além de normalizar o conteúdo textual, o pré-processamento também é responsável pela definição do vocabulário de termos que será utilizado na indexação e na construção de representações computacionais. Um bom pré-processamento reduz o tamanho do vocabulário, melhora o desempenho e aumenta a precisão das buscas (MANNING; RAGHAVAN; SCHüTZE, 2008). Essa etapa funciona, portanto, como uma ponte entre a linguagem natural e os algoritmos, permitindo que informações relevantes sejam extraídas de forma estruturada, eficiente e precisa. No contexto deste trabalho, ela se mostra essencial para a identificação de elementos fundamentais nos documentos jurídicos, como nomes das partes, dispositivos legais e decisões.

2.2.1 Tokenização

No contexto do processamento de linguagem natural, um *token* é definido como uma unidade básica de texto, normalmente correspondente a uma palavra, número, pontuação ou símbolo significativo. A forma como os *tokens* são extraídos influencia diretamente a eficácia dos modelos computacionais, especialmente em contextos especializados como o jurídico, onde há presença de construções formais e terminologias específicas.

O processo de tokenização refere-se, portanto, à separação do texto contínuo em unidades menores, os *tokens*, que geralmente representam palavras. Essa tarefa vai além de simplesmente dividir o texto por espaços; por exemplo, expressões como "São Paulo" ou "Art. 5º" exigem regras específicas para não serem segmentadas incorretamente, especialmente em textos técnicos como os jurídicos (MANNING; RAGHAVAN; SCHüTZE, 2008).

Figura 1 – Exemplo de *tokenização*.

```
>>> text = 'That U.S.A. poster-print costs $12.40...'
>>> pattern = r'''(?x)    # set flag to allow verbose regexps
...     (?:[A-Z]\.)+        # abbreviations, e.g. U.S.A.
...   | \w+(?:-\w+)*        # words with optional internal hyphens
...   | \$?\d+(?:\.\d+)?%?  # currency, percentages, e.g. $12.40, 82%
...   | \.\.\.              # ellipsis
...   | [][.,;"'?():_`-]  # these are separate tokens; includes ], [
... '''
>>> nltk.regexp_tokenize(text, pattern)
['That', 'U.S.A.', 'poster-print', 'costs', '$12.40', '...']
```

Fonte: (JURAFSKY; MARTIN, 2025)

2.2.1.1 Desafios específicos no contexto jurídico em português

A tokenização de textos jurídicos em língua portuguesa apresenta desafios particulares que precisam ser considerados para evitar a perda de informação ou a distorção semântica. Um dos principais aspectos envolve o tratamento da pontuação. Embora seja comum a remoção de sinais de pontuação na maioria dos sistemas de pré-processamento textual, certos elementos, como os números de processo — por exemplo, 1234567-89.2023.8.26.0100 — devem ser preservados como um único *token*, dada sua importância semântica e recorrência em documentos legais (ZAMPIERI et al., 2021).

Outro desafio é o manejo de contrações e apóstrofos. Expressões como *d'água* exigem normalização ou adaptação conforme a estratégia do sistema de NLP adotado, a fim de evitar a fragmentação de termos relevantes ou a exclusão de significados culturalmente marcados. Esse tipo de problema é particularmente sensível em português, dada a frequência de contrações e formas elípticas presentes na linguagem formal e jurídica (ZAMPIERI et al., 2021).

A presença de hífen também demanda atenção, especialmente em palavras compostas. Termos como *auto-aplicável* podem ser mantidos como uma única unidade ou separados, dependendo do contexto e do domínio de aplicação. Em geral, recomenda-se a separação de prefixos curtos em termos técnicos, mas a preservação do hífen em nomes próprios, como em *Hewlett-Packard*. Esse problema é bem documentado tanto na literatura geral de processamento de linguagem quanto nos trabalhos específicos aplicados ao domínio jurídico brasileiro, que destacam como a segmentação incorreta impacta diretamente na qualidade da indexação e da recuperação de informação (ZAMPIERI et al., 2021).

Além disso, há a necessidade de identificar e manter unidas expressões fixas e nomes compostos, tais como *Ministério Público*, *Poder Judiciário* ou *São Paulo*. Essas unidades semânticas são especialmente relevantes quando se lida com consultas baseadas em frases, nas quais a separação pode comprometer a precisão dos resultados. Modelos jurídicos treinados para o português, como os apresentados no projeto NLP Legal Brasil, reforçam a necessidade desse tipo de pré-processamento especializado, visto que expressões compostas carregam um valor semântico muito superior ao de seus termos isolados (ZAMPIERI et al., 2021).

Por fim, entidades estruturadas como endereços de e-mail, URLs, datas e identificadores numéricos — incluindo CPF e CNPJ — devem ser reconhecidas e tratadas como *tokens* indivisíveis. Esses elementos são frequentemente objeto direto de busca em sistemas jurídicos informatizados, e sua fragmentação pode prejudicar seriamente a eficácia dos mecanismos de recuperação de informação. A literatura destaca que a preservação de entidades estruturadas é uma das práticas essenciais para garantir a robustez de sistemas de busca no domínio jurídico, especialmente em bases documentais complexas como acervos de jurisprudência ou diários oficiais (ZAMPIERI et al., 2021).

2.2.2 Remoção de Stopwords

A remoção de *stopwords* é uma das etapas principais do pré-processamento textual e consiste na exclusão de palavras que ocorrem com alta frequência, mas que carregam pouco ou nenhum valor semântico individual — como *de*, *que*, *em*, *o* e *a*. Esse tipo de filtragem é comumente utilizado para reduzir o vocabulário e acelerar os algoritmos, sem perda significativa de informação relevante para a maioria das tarefas de recuperação e

classificação textual. (MANNING; RAGHAVAN; SCHüTZE, 2008)

Apesar de a remoção de *stopwords* ser uma prática comum em tarefas de processamento de linguagem natural, no contexto jurídico é necessário cautela. Isso porque palavras funcionais — geralmente ignoradas em outros domínios — podem conter informações fundamentais para a interpretação de dispositivos legais, como apontado por ZAMPIERI et al. (2021), que optaram por não removê-las durante o treinamento de seus modelos linguísticos, justamente para preservar a integridade semântica dos textos judiciais.

Figura 2 – Exemplo de remoção de *stopwords*

Sample text with Stop Words	Without Stop Words
GeeksforGeeks – A Computer Science Portal for Geeks	GeeksforGeeks , Computer Science, Portal ,Geeks
Can listening be exhausting?	Listening, Exhausting
I like reading, so I read	Like, Reading, read

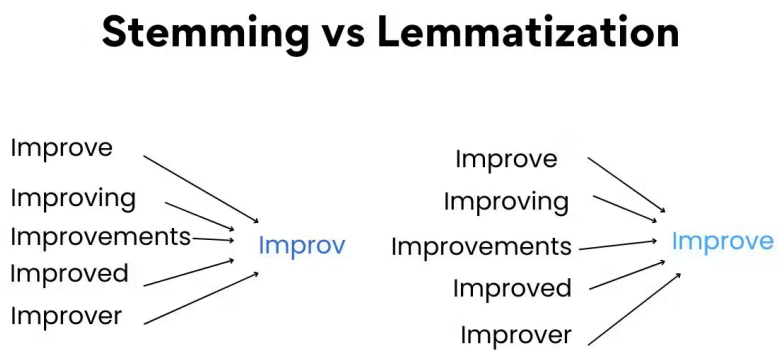
Fonte: (GEEKSFORGEEKS, 2024)

2.2.3 Lematização e Stemização

A lematização reduz palavras à sua forma canônica, conhecida como *lema*. Por exemplo, termos como *decidir*, *decisão* e *decidido* são todos associados ao verbo *decidir*. Esse processo considera a análise morfológica e o contexto gramatical, sendo essencial para manter o sentido das palavras nos textos jurídicos, onde pequenas variações morfológicas podem alterar significativamente o significado (JURAFSKY; MARTIN, 2025).

Por outro lado, a stemização é uma técnica mais simples que remove afixos (prefixos e sufixos) para obter uma raiz. No entanto, essa raiz nem sempre corresponde a uma palavra válida no idioma. Por exemplo, as palavras *decidir*, *decidindo* e *decidido* podem ser reduzidas a *decid*.

Figura 3 – Exemplo de diferenças entre Stemização e Lematização.



Fonte: (TURING, 2023)

A lematização proporciona maior fidelidade semântica, sendo mais indicada para tarefas que exigem compreensão precisa dos textos, como análise e busca em documentos jurídicos. Essa abordagem, por depender de análise linguística completa e regras morfológicas, costuma ter maior custo computacional e necessidade de recursos linguísticos bem estruturados, especialmente em idiomas como o português, que possui alta complexidade morfológica (ZAMPIERI et al., 2021).

Por outro lado, a stemização é uma solução mais simples e eficiente em termos computacionais, baseada em regras heurísticas para remoção de afixos. Embora menos precisa, pode ser útil em tarefas onde a sensibilidade semântica não é tão crítica (TURING, 2023).

No contexto jurídico brasileiro, a adoção de lematização se mostra mais vantajosa, uma vez que ferramentas linguísticas genéricas não são suficientes para capturar as nuances da linguagem jurídica em português (ZAMPIERI et al., 2021).

2.2.4 Segmentação de Sentenças

A segmentação de sentenças consiste em dividir um texto contínuo em unidades menores, geralmente delimitadas por frases. Esse processo é essencial para permitir que modelos de processamento de linguagem operem com unidades textuais semanticamente completas, sobretudo em tarefas que requerem maior nível de precisão, como extração de fundamentos jurídicos, sumarização automática, análise de jurisprudências ou geração de relatórios.

Embora seja considerado um processo relativamente simples no processamento de linguagem natural, a segmentação de sentenças apresenta desafios em textos jurídicos. A alta frequência de abreviações, siglas e expressões formais, como “Art.”, “Inc.”, “V. Exa.” ou “D.J.E.”, gera ambiguidades na definição dos pontos que efetivamente representam o final de uma sentença. Pontuações como ponto final, dois-pontos e até ponto e vírgula podem assumir funções distintas dependendo do contexto (ZAMPIERI et al., 2021).

2.3 Representação de Documentos

2.3.1 TF-IDF

Após o pré-processamento, uma das técnicas mais consolidadas para representar documentos como vetores numéricos é o TF-IDF (*Term Frequency–Inverse Document Frequency*). Esse método associa a cada termo um peso que reflete tanto sua importância local — baseada na frequência com que aparece no documento — quanto sua importância global — determinada pela raridade do termo na coleção como um todo (JURAFSKY; MARTIN, 2025; MANNING; RAGHAVAN; SCHüTZE, 2008).

O cálculo do TF–IDF é composto por duas etapas fundamentais. A primeira consiste na determinação da frequência de termo *Term Frequency* (TF), que quantifica o número de vezes que um termo t aparece em um documento d, representado por $tf_{t,d}$. Para evitar que termos muito frequentes dominem a representação, especialmente em documentos longos, é comum aplicar uma normalização logarítmica, definida como:

$$tf_{t,d} = \begin{cases} 1 + \log(tf_{t,d}), & \text{se } tf_{t,d} > 0, \\ 0, & \text{caso contrário.} \end{cases}$$

Essa transformação suaviza o impacto de termos com alta contagem, tornando a representação mais robusta e comparável entre documentos de diferentes tamanhos (MANNING; RAGHAVAN; SCHüTZE, 2008).

A segunda etapa envolve o cálculo da *Inverse Document Frequency* (IDF), que mede a capacidade de um termo discriminar documentos. Dado N como o número total de documentos na coleção e df_t como o número de documentos em que o termo t aparece, a IDF é calculada como:

$$idf_t = \log\left(\frac{N}{df_t}\right)$$

Esse fator tem como objetivo reduzir o peso de termos comuns na coleção, como artigos, pronomes ou palavras genéricas, e aumentar o peso de termos mais raros, que são mais úteis para diferenciar documentos (MANNING; RAGHAVAN; SCHüTZE, 2008).

Assim, o peso final de um termo em um documento, chamado de TF–IDF, é dado pela multiplicação dos dois componentes:

$$\text{tf-idf}(t, d) = tf_{t,d} \times idf_t$$

Esse valor será mais alto para termos que ocorrem frequentemente em um documento específico, mas que são raros na coleção como um todo, refletindo sua importância tanto local quanto global (MANNING; RAGHAVAN; SCHüTZE, 2008).

A representação de documentos no modelo vetorial é então construída a partir dos pesos TF–IDF de cada termo. Formalmente, um documento é representado como um vetor $\mathbf{v}(d) = [\text{tf-idf}(t_1, d), \ldots, \text{tf-idf}(t_M, d)]$, onde M é o número total de termos únicos na coleção. Consultas podem ser representadas de maneira análoga, permitindo o uso de métricas como a similaridade do cosseno para calcular a proximidade entre consultas e documentos no espaço vetorial (MANNING; RAGHAVAN; SCHüTZE, 2008).

Figura 4 – Exemplo da aplicação do TF-IDF em frases.

Index →	0	1	2	3	4	5	6	7	8
	and	**document**	**first**	**is**	**one**	**second**	**the**	**third**	**this**
"This is the first document."	0	0.46979139	0.58028582	0.38408524	0	0	0.38408524	0	0.38408524
"This document is the second document."	0	0.6876236	0	0.28108867	0	0.53864762	0.28108867	0	0.28108867
"And this is the third one."	0.51184851	0	0	0.26710379	0.51184851	0	0.26710379	0.51184851	0.26710379
"Is this the first document?"	0	0.46979139	0.58028582	0.38408524	0	0	0.38408524	0	0.38408524

Fonte: (RAHUL, 2023)

O modelo TF–IDF é amplamente utilizado em tarefas de recuperação de informação, classificação e agrupamento de textos, justamente por sua simplicidade e eficácia. Seu principal mérito reside na combinação da frequência local com a raridade global dos termos, permitindo destacar palavras que são relevantes para o conteúdo específico de um documento, enquanto penaliza termos genéricos e pouco informativos (JURAFSKY; MARTIN, 2025).

2.3.2 BM25

Enquanto o TF-IDF estabeleceu um modelo fundamental para a recuperação de informação, ele possui limitações intrínsecas, como a tendência de favorecer desproporcionalmente termos de alta frequência e a falta de uma normalização robusta para o tamanho dos documentos. Para superar essas questões, foi desenvolvido o algoritmo Okapi BM25 (doravante chamado apenas de BM25), um modelo de ranqueamento probabilístico que se tornou um padrão de fato em sistemas de busca modernos devido à sua eficácia e robustez (MANNING; RAGHAVAN; SCHüTZE, 2008).

O BM25 não trata a frequência de um termo (*term frequency*) de forma linear. Em vez disso, ele introduz um componente de saturação, partindo da premissa de que a relevância de um termo para um documento não cresce infinitamente com sua frequência. A primeira vez que uma palavra aparece é muito significativa; a décima, um pouco menos; e a centésima vez adiciona uma relevância marginal muito pequena. O algoritmo modela esse comportamento para evitar que documentos longos e repetitivos dominem os resultados da busca.

A pontuação de relevância de um documento D para uma consulta Q, que contém os termos $q_1, q_2, \ldots, q_n$, é calculada pela seguinte fórmula:

$$\text{score}(D, Q) = \sum_{i=1}^{n} \text{IDF}(q_i) \cdot \frac{f(q_i, D) \cdot (k_1 + 1)}{f(q_i, D) + k_1 \cdot \left(1 - b + b \cdot \frac{|D|}{\text{avgdl}}\right)}$$

Para entender seu funcionamento, podemos decompor a fórmula em suas três partes principais. O primeiro componente é o *Inverse Document Frequency* (IDF), que, assim como no TF-IDF, mede a raridade de um termo na coleção de documentos. O BM25 utiliza uma variante da fórmula padrão de IDF, geralmente com um ajuste para evitar valores nulos ou negativos para termos que aparecem em mais da metade dos documentos (MANNING; RAGHAVAN; SCHüTZE, 2008). A lógica central, no entanto, permanece a mesma: termos que são raros em toda a coleção recebem um peso maior, pois são considerados mais discriminativos.

O segundo e mais inovador componente da fórmula lida com a *saturação da frequência do termo*. Diferente do TF-IDF, o BM25 parte do princípio de que a relevância de um termo não cresce infinitamente com sua frequência. A primeira ocorrência é muito significativa, mas cada aparição subsequente contribui progressivamente menos para a pontuação final. Esse efeito de saturação é controlado pelo parâmetro de calibração *k1* (geralmente entre 1.2 e 2.0), que define quão rapidamente a pontuação de um termo se estabiliza. Um valor de *k1* baixo faz com que a relevância sature rapidamente, enquanto um valor mais alto permite que a frequência do termo continue a ter um impacto maior na pontuação (MANNING; RAGHAVAN; SCHüTZE, 2008).

Por fim, o BM25 introduz uma *normalização pelo comprimento do documento* muito mais sofisticada. O algoritmo compara o tamanho do documento atual, $|D|$, com o tamanho médio de todos os documentos na coleção, *avgdl*. O parâmetro b (geralmente em torno de 0.75) controla o grau de influência que o tamanho do documento tem na pontuação final. Quando b é 1, o efeito da normalização é máximo; quando é 0, o tamanho do documento é completamente ignorado. Essa abordagem permite penalizar documentos que são muito mais longos que a média, pois eles têm uma probabilidade estatisticamente maior de conter os termos da busca por acaso, sem serem necessariamente mais relevantes (MANNING;

RAGHAVAN; SCHüTZE, 2008).

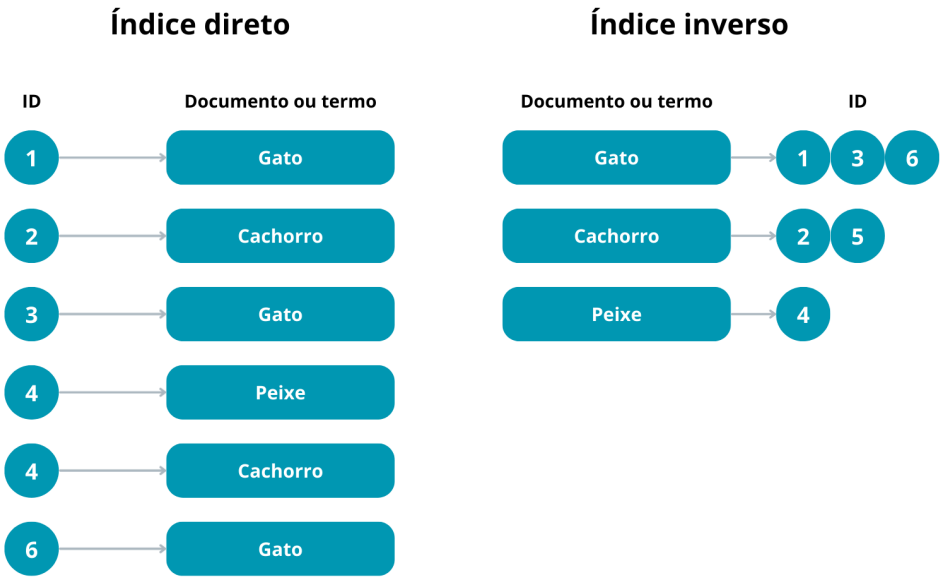
Em síntese, o BM25 aprimora os conceitos do modelo vetorial ao incorporar uma visão probabilística e heurísticas mais refinadas sobre como a frequência de termos e o tamanho dos documentos influenciam a relevância. Sua capacidade de ser calibrado pelos parâmetros *k1* e *b* o torna altamente adaptável a diferentes tipos de coleções de texto. Essa relevância é comprovada por sua aplicação prática em diversos domínios, incluindo estudos recentes no contexto jurídico brasileiro para a identificação de similaridade em documentos oficiais (RODRIGUES, 2024).

2.3.3 Índice inverso

O cálculo eficiente de medidas como o TF-IDF e o BM25, bem como a execução de buscas rápidas em grandes volumes de texto, dependem de uma estrutura de dados mais sofisticada do que a simples lista de documentos. A cada nova consulta ou para a construção dos próprios modelos estatísticos, seria computacionalmente inviável reanalisar todo o corpus para encontrar termos, suas frequências (*term frequency*) e em quantos documentos eles aparecem (*document frequency*). Para resolver esse problema, utiliza-se o **índice inverso** (ou *inverted index*), que funciona como a principal estrutura de indexação para sistemas de recuperação de informação (MANNING; RAGHAVAN; SCHüTZE, 2008).

O índice inverso é uma estrutura de dados amplamente utilizada em sistemas de recuperação de informação, projetada para otimizar a busca de termos em grandes coleções de documentos (MANNING; RAGHAVAN; SCHüTZE, 2008; ALMEIDA; DIAS, 2023). Diferentemente de uma abordagem direta, que examinaria cada documento sequencialmente (uma operação de complexidade $O(N)$, onde N é o tamanho do corpus), o índice invertido organiza os termos do corpus de forma a mapear cada palavra ou unidade lexical às suas respectivas ocorrências. Esse mapeamento inclui informações como a identificação dos documentos e, comumente, a posição ou frequência de cada termo. Essa organização permite consultas eficientes e rápidas, fundamentais para o funcionamento de sistemas como motores de busca (MANNING; RAGHAVAN; SCHüTZE, 2008). Segue uma imagem que exemplifica o índice inverso.

Figura 5 – No índice inverso quem mapeia são os termos e n.



Fonte: Elaboração própria

Uma forma de otimizar o uso de memória do índice inverso é por meio da compressão das listas de ocorrências (*postings lists*). Técnicas comuns incluem a codificação de diferenças (*gap encoding* ou *delta encoding*), onde apenas a diferença entre IDs de documentos consecutivos é armazenada, e codificações de inteiros como *varint* (*Variable Byte coding*) ou *Golomb coding*, que reduzem o espaço ocupado por números pequenos. Além de economizar memória, a compressão pode acelerar consultas, já que menos dados precisam ser carregados da memória ou disco, sendo uma prática padrão em motores de busca de larga escala (MANNING; RAGHAVAN; SCHüTZE, 2008).

O índice inverso oferece consultas com complexidade $O(\log n)$ para a busca de termos no dicionário da estrutura, onde n é o número de termos únicos, caso o dicionário seja implementado com árvores balanceadas; ou $O(1)$ em média, se forem usadas tabelas hash (MANNING; RAGHAVAN; SCHüTZE, 2008). Isso se difere drasticamente da complexidade $O(N)$ de uma varredura sequencial. Atualizações, como a inserção de novos documentos, têm um custo associado ao processamento de seus termos. No projeto, usamos uma tabelas hash para mapear termos a listas de ocorrências, armazenando para cada termo: ID do documento, frequência e posições. Isso reduz o tempo de vetorização, já que os dados são pré-computados, garantindo eficiência em consultas e escalabilidade para corpora grandes.

O índice inverso, contudo, apresenta *trade-offs*. Ele consome mais memória do que uma varredura direta, devido ao armazenamento do dicionário de termos e das listas de ocorrências. A construção inicial do índice também é um processo custoso, com complexidade tipicamente linear em relação ao tamanho total do corpus (MANNING; RAGHAVAN; SCHüTZE, 2008). Em corpora dinâmicos, onde documentos são adicionados ou alterados frequentemente, as atualizações podem exigir estratégias complexas de gerenciamento,

como o uso de índices auxiliares ou a fusão periódica de índices incrementais, para manter o desempenho sem reconstruir toda a estrutura (MANNING; RAGHAVAN; SCHüTZE, 2008).

2.3.4 *Embeddings* baseados em Inteligência Artificial

Modelos estatísticos clássicos de recuperação de informação, como o TF-IDF e o BM25, operam em um nível lexical. Eles são extremamente eficientes para determinar a relevância de um documento com base na frequência e raridade de *palavras-chave* exatas, mas falham em capturar o *significado* ou a *intenção* por trás da busca. Por exemplo, uma consulta por "responsabilidade civil em acidentes de trânsito"pode não retornar um documento altamente relevante que use a frase "indenização por colisão de veículos", pois os termos não coincidem. Essa limitação, conhecida como *impedância semântica* (*semantic mismatch*), é o principal desafio que as representações de texto baseadas em Inteligência Artificial buscam resolver.

Para superar essa barreira, o Processamento de Linguagem Natural (NLP) evoluiu de modelos baseados em contagem para modelos baseados em *predição*, que aprendem a representar palavras em um espaço vetorial contínuo. Essas representações, conhecidas como *word embeddings* (ou apenas *embeddings*), são vetores densos de números que capturam relações semânticas e sintáticas complexas a partir do contexto em que as palavras aparecem em grandes volumes de texto (MIKOLOV KAI CHEN; DEAN, 2013). Em vez de apenas contar palavras, esses modelos aprendem seu significado, impulsionando uma nova geração de sistemas de busca semântica.

2.3.4.1 Processamento de Linguagem Natural (NLP)

O Processamento de Linguagem Natural (NLP) é uma área da inteligência artificial que tem como objetivo permitir que computadores compreendam, interpretem e gerem a linguagem humana de forma automática e eficiente. A linguagem natural, usada por pessoas no dia a dia, é cheia de ambiguidades, variações e contextos, o que torna o trabalho do NLP bastante desafiador. Para superar essas dificuldades, o NLP aplica técnicas que vão desde regras gramaticais, análise sintática, tokenização e stemming até modelos probabilísticos, aprendizado supervisionado, redes neurais e arquiteturas avançadas como *transformers* (SEZERER, 2021).

Historicamente, os sistemas de NLP eram baseados em regras linguísticas definidas manualmente, o que exigia muito esforço e apresentava limitações para lidar com a complexidade da linguagem. Com o avanço do aprendizado de máquina, especialmente dos modelos de aprendizado profundo, o campo evoluiu significativamente. Modelos modernos, como os baseados em arquiteturas *transformers*, têm a capacidade de capturar relações contextuais complexas em textos, o que melhorou muito o desempenho em tarefas essenciais do NLP, como tradução automática, análise de sentimentos, reconhecimento de fala, extração de informações, sumarização automática e respostas a perguntas (REIMERS, 2019). Essas técnicas permitem que sistemas interpretem o significado de frases, reconheçam intenções e até gerem textos coerentes, facilitando a interação entre humanos e máquinas.

O NLP precisa lidar com desafios como a ambiguidade das palavras — onde uma mesma palavra pode ter vários significados dependendo do contexto —, a variação linguística entre diferentes regiões e falantes, e a necessidade de entender o contexto mais

amplo para interpretar corretamente expressões e intenções. Além disso, o desenvolvimento de sistemas eficientes depende da disponibilidade de grandes volumes de dados anotados para treinar esses modelos, o que nem sempre é fácil para línguas menos representadas (SEZERER, 2021).

Na prática, o NLP está presente em muitas aplicações que usamos diariamente, desde assistentes virtuais como Siri e Alexa, passando por sistemas de tradução automática, *chatbots* para atendimento ao cliente, até ferramentas que ajudam a analisar opiniões em redes sociais. Essa popularização faz do NLP uma área essencial para a interação entre humanos e máquinas, impulsionando a transformação digital em diversos setores (REIMERS, 2019).

Nesse contexto de evolução das técnicas de NLP, um dos avanços mais importantes para a representação da linguagem foi o desenvolvimento dos *embeddings*. Eles surgiram como uma solução eficaz para transformar palavras, frases ou documentos em vetores numéricos que preservam relações semânticas e contextuais (SEZERER, 2021; MIKOLOV KAI CHEN; DEAN, 2013). Essa representação vetorial tornou-se fundamental para alimentar modelos de aprendizado de máquina, especialmente os modelos mais modernos. A seguir, serão explorados o conceito de *embeddings* e os principais modelos utilizados para gerá-los.

2.3.4.2 O que são *embeddings*

No contexto do Processamento de Linguagem Natural, *embeddings* são representações vetoriais densas de palavras, sentenças ou documentos, projetadas para capturar aspectos semânticos da linguagem em um espaço numérico contínuo. Diferente de representações tradicionais esparsas, como *one-hot encoding* ou a própria matriz TF-IDF, que apenas indicam a presença ou a frequência de palavras, os *embeddings* posicionam elementos linguisticamente semelhantes mais próximos entre si em um espaço vetorial de centenas ou milhares de dimensões (MIKOLOV KAI CHEN; DEAN, 2013). Isso permite que modelos matemáticos e algoritmos de aprendizado de máquina consigam processar e "entender"a linguagem de forma mais contextualizada e eficiente.

Essas representações são aprendidas automaticamente a partir de grandes volumes de dados textuais e permitem que relações semânticas e sintáticas emergentes sejam capturadas de maneira distribuída. Por exemplo, vetores de palavras como “rei” e “rainha” costumam estar próximos no espaço de *embeddings*, e relações analógicas como “rei – homem + mulher = rainha” podem ser observadas em modelos bem treinados (MIKOLOV KAI CHEN; DEAN, 2013). Essa capacidade de capturar padrões complexos tornou os *embeddings* um componente essencial em praticamente todas as aplicações modernas de NLP (PENNINGTON RICHARD SOCHER, 2014).

2.3.4.3 A Evolução dos Modelos de *Embeddings*

Os modelos para geração de *embeddings* evoluíram significativamente, passando de representações estáticas para representações dinâmicas e contextuais.

2.3.4.3.1 Embeddings Estáticos: Word2Vec e GloVe

Os primeiros modelos que popularizaram os *embeddings* foram o *Word2Vec* (MIKOLOV KAI CHEN; DEAN, 2013) e o *GloVe* (PENNINGTON RICHARD SOCHER,

2014). O *Word2Vec* é baseado em aprendizado preditivo local, utilizando duas arquiteturas principais: *Skip-Gram* e *Continuous Bag-of-Words* (CBOW). O modelo *Skip-Gram* tenta prever as palavras de contexto (vizinhas) a partir de uma palavra central. Inversamente, o CBOW tenta prever a palavra central com base em seu contexto. Por outro lado, o *GloVe* adota uma abordagem baseada em contagem, construindo vetores a partir de estatísticas globais de coocorrência de palavras em todo o corpus.

A principal característica — e limitação — desses modelos é que eles geram *embeddings estáticos*. Cada palavra no vocabulário é mapeada para um único vetor, independentemente do contexto em que ela é usada. Isso significa que a palavra "banco"teria a mesma representação vetorial em "sentei no banco da praça"e em "fui ao banco depositar dinheiro". Essa incapacidade de lidar com a polissemia (múltiplos significados) foi a principal motivação para o desenvolvimento da próxima geração de modelos.

2.3.4.3.2 Embeddings Contextuais: BERT e a Revolução *Transformer*

A grande revolução nos *embeddings* veio com os modelos de linguagem contextualizados, como o ELMo (PETERS MARK NEUMANN et al., 2018) e, principalmente, o BERT (DEVLIN MING-WEI CHANG; TOUTANOVA, 2019). Esses modelos não atribuem um vetor fixo a uma palavra, mas geram um *embedding* dinâmico para cada palavra com base na sentença completa em que ela aparece. A palavra "banco"passa a ter vetores diferentes e apropriados para cada um dos seus significados.

Essa capacidade foi possibilitada pela arquitetura *Transformer*, que introduziu um mecanismo chamado **autoatenção** (*self-attention*). Diferente de modelos anteriores que liam o texto sequencialmente (da esquerda para a direita), o mecanismo de autoatenção permite que o modelo "olhe"para todas as outras palavras da sentença simultaneamente, ponderando a importância de cada uma para definir o significado da palavra atual.

O BERT (*Bidirectional Encoder Representations from Transformers*) utiliza essa arquitetura de forma "bidirecional", lendo todo o contexto de uma vez. Para alcançar essa compreensão profunda, o BERT é pré-treinado em duas tarefas principais:

1. **Masked Language Model (MLM):** O modelo recebe uma sentença onde algumas palavras foram substituídas por um *token* especial ‘[MASK]‘. A tarefa do modelo é adivinhar qual era a palavra original, forçando-o a entender o contexto gramatical e semântico de ambos os lados (esquerdo e direito) da palavra mascarada.

2. **Next Sentence Prediction (NSP):** O modelo recebe dois segmentos de texto, A e B, e deve prever se B é a sentença que realmente se segue a A no texto original ou se é uma sentença aleatória do corpus. Isso ensina o modelo a entender a relação lógica e coesiva entre sentenças.

O resultado desse pré-treinamento intensivo é um modelo capaz de gerar *embeddings* contextuais de alta fidelidade, que capturam nuances da linguagem. Modelos subsequentes, como o Sentence-BERT (REIMERS, 2019), foram desenvolvidos especificamente para otimizar o BERT para tarefas de comparação de sentenças, gerando representações vetoriais de frases inteiras, ideais para busca semântica.

2.3.4.4 Recuperação e Medidas de Similaridade

A recuperação de informações em contextos de linguagem natural depende da capacidade que temos de medir similaridade entre todas estas representações textuais. Com o uso de *embeddings*, textos são convertidos em vetores numéricos em espaços de alta dimensão, nos permitindo aplicar técnicas matemáticas e equações para comparar conteúdos. Entre as diversas medidas de similaridade existentes — como a distância euclidiana, de *Jaccard* ou de *Manhattan* — a similaridade de cosseno destaca-se por sua robustez e simplicidade, sendo amplamente adotada em tarefas de busca semântica e ranqueamento de documentos. Na subseção a seguir, detalha-se seu funcionamento e aplicação no contexto de NLP e jurídico.

2.3.4.4.1 Similaridade de cosseno

A similaridade de cosseno é uma métrica amplamente utilizada em Processamento de Linguagem Natural (NLP) para medir o grau de similaridade entre dois vetores em um espaço vetorial multidimensional. Quando aplicada a *embeddings* de palavras, frases ou documentos, essa métrica permite quantificar o quão semanticamente próximos dois textos são entre si. A ideia central é comparar a orientação dos vetores e não sua magnitude, o que torna essa medida especialmente útil para dados textuais (MANNING; RAGHAVAN; SCHüTZE, 2008).

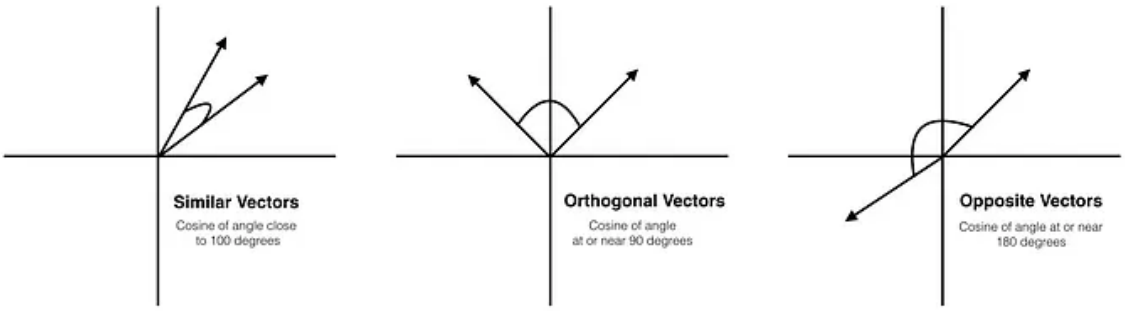
A fórmula da similaridade de cosseno entre dois vetores $\vec{A}$ e $\vec{B}$ é dada por:

$$\text{similaridade}_{\cos}(\vec{A}, \vec{B}) = \frac{\vec{A} \cdot \vec{B}}{||\vec{A}|| \cdot ||\vec{B}||}$$

onde $\vec{A} \cdot \vec{B}$ é o produto escalar entre os vetores, e $||\vec{A}||$ e $||\vec{B}||$ são as normas (módulos) dos respectivos vetores. O resultado varia entre -1 e 1, mas no contexto de NLP, os valores geralmente estão entre 0 (sem similaridade) e 1 (similaridade máxima), já que os vetores de *embeddings* costumam estar em um espaço de dimensão positiva.

O resultado da similaridade de cosseno varia entre -1 e 1, pois esse é o intervalo natural da função cosseno. Valores próximos de 1 indicam que os vetores estão quase na mesma direção, sugerindo alta similaridade. Se o valor estiver perto de 0, significa que os vetores são ortogonais, ou seja, não têm relação direta. Já valores próximos de -1 indicam direções opostas, o que representa dissimilaridade, como mostra a imagem abaixo em uma simplificação 2D:

Figura 6 – O ângulo entre os vetores é diretamente proporcional à sua similaridade.



Fonte: (CONTRATRES, 2018)

A vantagem da similaridade cosseno sobre outras métricas, como a distância euclidiana, está no fato de que ela é invariável à magnitude dos vetores. Isso significa que dois vetores com direções similares, mas comprimentos diferentes (por exemplo, textos curtos vs. longos com mesmo conteúdo), ainda podem ser considerados semanticamente similares. Isso é particularmente útil em tarefas de busca semântica e recuperação de informações, onde a intenção é encontrar conteúdos com sentido próximo, independentemente da sua extensão textual (MANNING; RAGHAVAN; SCHüTZE, 2008).

Na prática, ao aplicar essa métrica, os textos são primeiro convertidos em vetores numéricos usando técnicas de *embeddings* (como Word2Vec, BERT ou Sentence-BERT), e depois esses vetores são comparados utilizando a similaridade cosseno para ranquear ou agrupar documentos. Essa abordagem é a base de diversos sistemas modernos de busca jurídica, recomendação de jurisprudência e agrupamento de decisões semelhantes (REIMERS, 2019).

É importante ressaltar que o cosseno não capta relações mais complexas como ironia, polissemia ou implicaturas contextuais — para esses casos, a escolha do modelo de *embedding* (estático ou contextualizado) é determinante para que os vetores usados na comparação contenham essas nuances semânticas. Ainda assim, a similaridade cosseno continua sendo uma escolha padrão e eficaz em aplicações de NLP por sua simplicidade, desempenho e capacidade de generalização (JURAFSKY; MARTIN, 2025).

2.3.4.5 *Embeddings* aplicados ao Direito

No contexto jurídico, *embeddings* vêm sendo utilizados para representar textos legais em formato vetorial, permitindo a aplicação de algoritmos de NLP em tarefas como busca jurisprudencial, classificação de documentos, extração de entidades e verificação de similaridade entre decisões. Ao converter petições, sentenças, leis e pareceres em vetores numéricos, torna-se possível aplicar modelos de aprendizado para identificar padrões, sugerir documentos similares ou agrupar textos por temas jurídicos (CHALKIDIS MANOS FERGADIOTIS; ALETRAS; ANDROUTSOPOULOS, 2020).

Uma das principais vantagens dos *embeddings* nesse domínio é a capacidade de lidar com a linguagem técnica e formal do Direito. Modelos genéricos, treinados com textos da web, podem não capturar as nuances de termos jurídicos. Por isso, modelos da família

LEGAL-BERT (CHALKIDIS MANOS FERGADIOTIS; ALETRAS; ANDROUTSOPOULOS, 2020), que são pré-treinados ou ajustados (*fine-tuned*) especificamente em corpora jurídicos, tornam-se mais eficazes na captação de vocabulário especializado e estruturas discursivas recorrentes. Contudo, o treinamento e a inferência desses modelos demandam alto custo computacional, o que levanta questões sobre o *trade-off* entre precisão semântica e eficiência, cerne da investigação deste trabalho (REIMERS, 2019; COSTA, 2023).

2.4 Algoritmos

Algoritmos são conjuntos finitos de instruções bem definidas, ordenadas e executáveis que visam resolver um problema ou realizar uma tarefa específica. Conforme definido por Cormen e outros. (CORMEN et al., 2013), "um algoritmo é qualquer procedimento computacional bem definido que toma algum valor ou conjunto de valores como entrada e produz algum valor ou conjunto de valores como saída". Segundo o dicionário Michaelis (LTDA., 2025), algoritmo é "Conjunto de regras e operações e procedimentos, definidos e ordenados usados na solução de um problema". Dessa forma, os algoritmos constituem a base do raciocínio computacional e do desenvolvimento de software, estruturando a lógica para a execução de tarefas de maneira clara e eficiente.

2.4.1 Avaliação de algoritmos: Coeficiente de *Spearman*

Neste trabalho, após ordenarmos os documentos do mais ao menos importante com base nos algoritmos analisados, é fundamental dispor de uma métrica que nos permita comparar essas listas. Essa comparação é essencial para medir a consistência dos algoritmos e avaliar quão semelhantes são os rankings que eles produzem (MOTTA et al., 2020). Para isso, utilizaremos o coeficiente de correlação de *Spearman*, uma técnica estatística não paramétrica que quantifica a força e a direção da relação entre duas variáveis ordenadas (SIEGEL; CASTELLAN N. JOHN, 1988).

Para entender o coeficiente de *Spearman*, é útil primeiro diferenciá-lo do coeficiente de correlação de *Pearson*. O *Pearson* é uma medida estatística que indica a força e a direção da relação *linear* entre duas variáveis contínuas. Ele avalia se os pontos de dados se aproximam de uma linha reta, mas é sensível a *outliers* e assume que os dados seguem uma distribuição normal (SIEGEL; CASTELLAN N. JOHN, 1988).

O *Pearson* avalia diretamente os valores dos *scores* de relevância, o que não é ideal para o nosso problema. Não nos importa se o *score* do BM25 foi 0.8 e o do *embedding* foi 0.9; o que nos importa é se ambos concordam que aquele documento é o "primeiro mais relevante". É justamente nesse cenário que o coeficiente de *Spearman* se torna mais apropriado. Ele não olha para os *scores* brutos, mas sim para os *postos* (ou *ranks*) dos elementos, medindo a correlação monotônica entre eles (SIEGEL; CASTELLAN N. JOHN, 1988; MOTTA et al., 2020).

Na prática, o coeficiente de *Spearman* é simplesmente definido como o **coeficiente de correlação de *Pearson* calculado sobre os postos das variáveis**. O processo de cálculo envolve transformar os *scores* de cada lista em uma sequência de postos (1º, 2º, 3º, ...). Caso ocorram empates (por exemplo, dois documentos com o mesmo *score*), atribui-se a eles a média de suas posições. Por exemplo, se dois documentos empatam nas posições 2 e 3, ambos recebem o posto 2,5. Após a transformação, o coeficiente de *Pearson* é aplicado sobre essas duas novas listas de postos (SIEGEL; CASTELLAN N. JOHN, 1988).

O resultado, assim como o *Pearson*, varia de -1 a +1. Valores próximos de +1 indicam uma forte concordância (os algoritmos produziram rankings muito semelhantes). Valores próximos de -1 indicam uma forte discordância (um algoritmo ranqueou na ordem inversa do outro). Valores próximos de 0 apontam para uma ausência de correlação, sugerindo que os rankings são aleatórios um em relação ao outro (SIEGEL; CASTELLAN N. JOHN, 1988).

No contexto deste trabalho, o coeficiente de *Spearman* fornecerá uma medida objetiva do grau de concordância entre as listas produzidas pelas abordagens estatísticas (TF-IDF, BM25) e semânticas (*embeddings*). Conforme demonstrado em Motta et al. (2020), que utilizou essa métrica para comparar algoritmos de ranqueamento textual, o *Spearman* é uma ferramenta eficaz para avaliar a correlação entre os resultados de diferentes sistemas de busca.

2.4.2 Algoritmo de busca de padrão: Boyer-Moore

O algoritmo Boyer-Moore é uma técnica avançada de busca de padrões em textos, criada em 1977 por Robert S. Boyer e J Strother Moore, dois cientistas da computação americanos. Eles desenvolveram esse método enquanto trabalhavam na IBM, buscando uma solução eficiente para o problema clássico de encontrar a ocorrência de uma sequência de caracteres (padrão) dentro de um texto maior, que é uma tarefa fundamental em várias áreas, como processamento de texto, bioinformática e sistemas de busca (BOYER, 1977; GUSFIELD, 1997).

O Boyer-Moore destaca-se por sua eficiência prática e seu funcionamento baseado em heurísticas inteligentes, que permitem ao algoritmo pular grandes trechos do texto durante a busca, ao invés de verificar cada caractere sequencialmente como em métodos mais simples. Isso o torna especialmente rápido para textos grandes e padrões não triviais (BOYER, 1977; HORSPOOL, 1980).

A ideia central do algoritmo é comparar o padrão com o texto da direita para a esquerda, começando pelo último caractere do padrão. Caso ocorra uma incompatibilidade, duas heurísticas são usadas para calcular possíveis deslocamentos do padrão: a heurística do caractere ruim (*Bad Character Rule*) e a heurística do sufixo bom (*Good Suffix Rule*) (GUSFIELD, 1997; BOYER, 1977).

A heurística do caractere ruim atua quando há uma falha na comparação entre o padrão e o texto, ou seja, quando um caractere do texto não coincide com o caractere correspondente do padrão. Nesse caso, o algoritmo verifica se esse caractere "ruim" aparece em alguma posição anterior dentro do próprio padrão. Se ele existir, o padrão é deslocado para a direita até que essa ocorrência do caractere ruim no padrão alinhe-se com a posição atual no texto. Caso o caractere não esteja presente no padrão, o deslocamento pode ser ainda maior, movendo o padrão além desse caractere, o que permite pular várias posições de uma vez (GUSFIELD, 1997; BOYER, 1977).

Já a heurística do sufixo bom é aplicada quando um sufixo do padrão já casou com uma parte do texto, mas a comparação subsequente falha. Nesse cenário, o algoritmo tenta alinhar esse sufixo com outra ocorrência dele dentro do próprio padrão, deslocando-o para a direita. Se não houver outra ocorrência desse sufixo, busca-se alinhar parte desse sufixo com um prefixo correspondente do padrão. Dessa forma, essa heurística permite que o algoritmo aproveite ao máximo as comparações já realizadas, promovendo deslocamentos

que aumentam a eficiência da busca sem risco de perder possíveis correspondências (GUSFIELD, 1997; BOYER, 1977).

Essas duas heurísticas trabalham juntas para maximizar o deslocamento do padrão e minimizar o número de comparações. Na prática, isso significa que o algoritmo consegue pular grandes partes do texto onde a busca não terá sucesso, evitando comparações desnecessárias. Por isso, seu desempenho médio é muito eficiente, com complexidade sublinear — ou seja, ele não precisa examinar todos os caracteres do texto. Essa eficiência se traduz em um tempo médio aproximado de $O\left(\frac{n}{m}\right)$, onde n é o tamanho do texto e m o tamanho do padrão, o que é muito melhor do que a abordagem ingênua que sempre leva $O(n \cdot m)$ (BOYER, 1977; GUSFIELD, 1997; HORSPOOL, 1980).

No pior caso, entretanto, o Boyer-Moore pode chegar a uma complexidade de $O(n \cdot m)$, especialmente em textos com padrões altamente repetitivos ou quando o padrão tem caracteres repetidos que causam pequenos deslocamentos a cada falha. Apesar disso, esses casos são raros, e na prática, o algoritmo é considerado um dos mais rápidos e robustos para busca de padrões em textos (BOYER, 1977; GALIL, 1979; GUSFIELD, 1997).

Em contrapartida, o algoritmo de KMP evita retrocessos no texto usando a tabela de falhas (prefix-função), garantindo sempre tempo linear $O(n + m)$, onde n é o tamanho do texto e m o do padrão. Já Boyer-Moore, embora não tenha garantia de pior caso linear sem extensões, costuma superar KMP em textos reais graças aos grandes saltos proporcionados pelas heurísticas de caractere-ruim e sufixo-bom (KNUTH JAMES HIRAM MORRIS JUNIOR, 1977; BOYER, 1977).

2.4.3 Algoritmos de Ordenação - Quicksort

O algoritmo *Quicksort* é um método de ordenação de listas criado em 1960 pelo britânico Charles Antony Richard Hoare, enquanto ainda era estudante em Moscou. A publicação oficial do algoritmo ocorreu em 1962. O *Quicksort* se baseia na técnica de divisão e conquista, utilizando recursão para quebrar o problema em partes menores, com complexidade média de $O(n \log n)$ e pior caso de $O(n^2)$, dependendo da escolha do pivô.

A técnica de divisão e conquista envolve três etapas fundamentais: dividir, resolver e combinar. Primeiro, o problema original é fragmentado em partes menores — de preferência equilibradas — reduzindo sua complexidade. Em seguida, cada subproblema é resolvido independentemente, geralmente por meio de chamadas recursivas. Por fim, os resultados são reunidos para formar a solução completa, permitindo que problemas grandes sejam solucionados por meio da repetição controlada de operações simples (CORMEN et al., 2013).

Essa abordagem parte da ideia de que resolver vários subproblemas simples costuma ser mais eficiente do que atacar o problema original diretamente. No *Quicksort*, isso se traduz em dividir uma lista de tamanho n em duas sublistas menores e aplicar o mesmo procedimento recursivo a cada uma. A recursão prossegue até que cada sublista tenha menos de dois elementos, sendo então considerada ordenada.

Hoare propôs que essa divisão fosse feita a partir da escolha arbitrária de um elemento da lista, chamado de pivô. Em seguida, percorre-se a lista comparando os elementos com o pivô: os menores são agrupados em uma nova lista, chamada vetor inferior (V_{inf}), e os maiores ou iguais, em outra, chamada vetor superior (V_{sup}).

Após o particionamento, obtemos três partes distintas: o vetor inferior (V_{inf}), o pivô central e o vetor superior (V_{sup}). O algoritmo então se auto-invoca recursivamente sobre V_{inf} e V_{sup}, até que as sublistas estejam suficientemente pequenas. Quando isso acontece, o *Quicksort* realiza a etapa final: a junção das três partes — V_{inf}, pivô e V_{sup} — reconstruindo a lista original, agora ordenada (ZIVIANI, 2004).

Figura 7 – Quicksort

```
fun (this Vector[T]) quickSort() Vector[T] = when {

    // Se o array tiver apenas um elemento, retorna ele mesmo
    this.size < 2 -> this

    // Caso contrário:
    else -> {
        // Pega o primeiro elemento
        val pivot = this.first()

        // Separa o array principal em duas partições para maiores e menores que o 'pivot'
        // ignorando o primeiro elemento pois este é o pivot
        val (vinf, vsup) = this.drop(1).partition { it <= pivot }

        // Retorna uma nova lista composta pelas chamadas recursivas ordenadas junto ao pivot
        vinf.quickSort() + pivot + vsup.quickSort()
    }
}
```

Fonte: Elaboração própria.

.

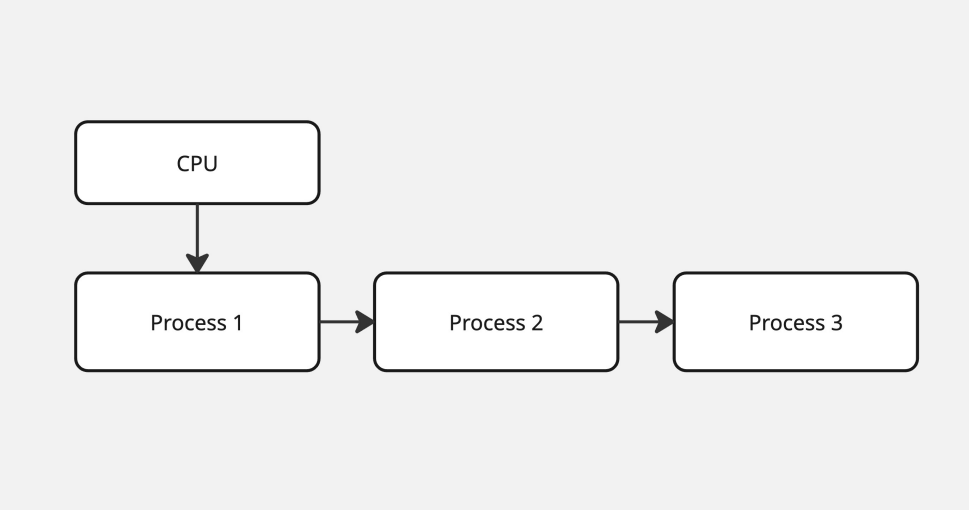
2.5 Programação Paralela e Escalabilidade

A programação paralela vem como uma das soluções propostas para os desafios modernos da computação. A ideia por trás dela foi impulsionada pela necessidade de otimizar o tempo de processamento através algoritmos eficientes para criar aplicações que sejam capazes de lidar com um grande volume de dados e cálculos intensivos (COELHO, 2015), coisa que os algoritmos sequenciais não são tão adequados.

2.5.1 Algoritmos Sequenciais e Limitações

Algoritmos sequenciais representam o início da computação, onde processadores e algoritmos tinham a premissa da restrição a ordem das instruções, ou seja, uma operação de cada vez, o fim de uma representa o início da próxima. Seguindo assim uma sequência lógica e clara (PAULA, 2019) como é representado na imagem abaixo.

Figura 8 – Algoritmos sequenciais

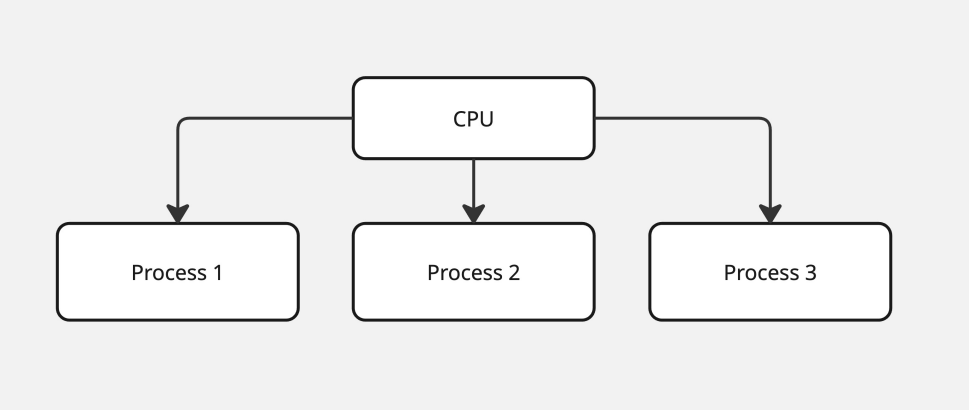


Fonte: Elaboração própria.

2.5.2 Algoritmos Paralelos: Modelos e Bibliotecas

Algoritmos paralelos são a forma de programação que se utiliza de arquiteturas paralelas e suas características para que várias operações e instruções possam ser executadas simultaneamente dentro da estrutura, podendo esta ser, *multicore*, multiprocessador ou uma rede de computadores (cluster) (BLELLOCH; MAGGS, 1996).

Figura 9 – Algoritmos paralelos



Fonte: Elaboração própria.

Com o avanço da computação, da engenharia de software, o surgimento de várias arquiteturas *multicore* e sistemas distribuídos, a paralelização dentro dessas estruturas nos permite dividir tarefas complexas em subtarefas executadas simultaneamente, maximizando a eficiência e o desempenho. Contudo, o desenvolvimento de algoritmos paralelos exige lidar com questões como sincronização, concorrência e balanceamento de carga, tornando essa prática essencial para atender às demandas de rapidez e escalabilidade atualmente (PAULA, 2019).

2.6 Trabalhos Relacionados

3 Desenvolvimento

O desenvolvimento deste trabalho seguirá uma abordagem incremental e iterativa, com entregas organizadas semanalmente ao longo de doze semanas. Cada etapa será planejada de forma a permitir a validação contínua dos componentes implementados, favorecendo a integração progressiva entre os módulos do sistema. Essa estratégia contribui para a identificação precoce de falhas e a mitigação de riscos relacionados a incompatibilidades entre partes distintas da aplicação.

A arquitetura adotada será orientada a microserviços, promovendo uma separação clara de responsabilidades entre os componentes do sistema. Essa escolha visa facilitar o desenvolvimento paralelo, a escalabilidade e a manutenção futura. O sistema terá como principal objetivo a extração e recuperação de informações contidas em documentos jurídicos, com foco em consultas tanto textuais quanto semânticas. A base de dados será composta por documentos oriundos de fontes públicas, como a Câmara dos Deputados, o Senado Federal e tribunais superiores. Esses documentos serão extraídos, normalizados e armazenados em um formato unificado, acompanhados de metadados que possibilitem buscas mais eficientes.

As buscas ocorrerão em duas camadas complementares. A camada textual utilizará o algoritmo de Boyer–Moore, reconhecido por seu bom desempenho na localização de padrões em grandes volumes de texto. Já a camada semântica aplicará modelos pré-treinados baseados em *transformers*, responsáveis por converter os textos em vetores numéricos. A partir desses vetores, será possível calcular a similaridade entre documentos com base na métrica de cosseno, promovendo uma recuperação mais contextualizada das informações.

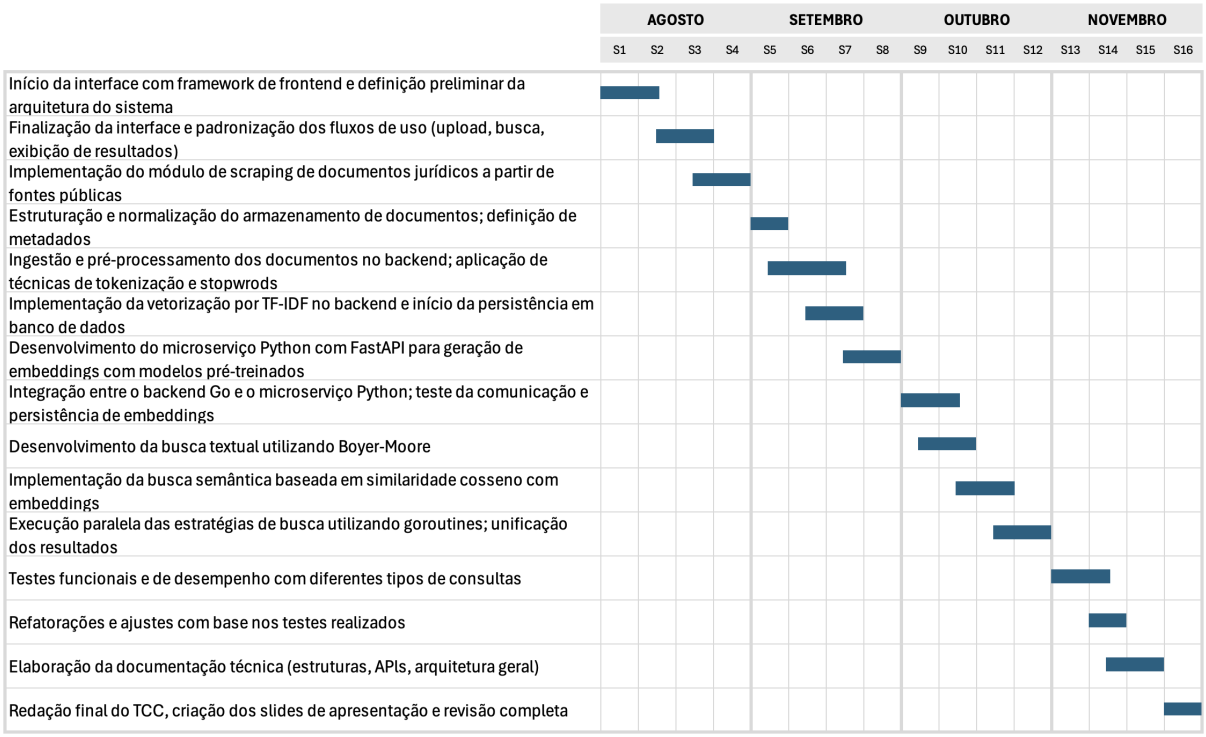
A infraestrutura do sistema será composta por três componentes principais, interconectados por meio de APIs REST. A camada de apresentação será construída com um *framework* moderno de *frontend* utilizando *TypeScript*, oferecendo ao usuário funcionalidades como envio de documentos, execução de buscas e visualização dos resultados. O *backend* principal será desenvolvido em GO (versão 1.22 ou superior), e será responsável pelas tarefas de ingestão dos documentos, normalização textual, armazenamento em banco de dados relacional e execução da lógica de busca.

O processamento semântico será realizado por um microserviço desenvolvido em Python 3.11, que utilizará a biblioteca *sentence-transformers* e modelos da família BERT, selecionados por apresentarem um bom equilíbrio entre desempenho e custo computacional. Esse microserviço será exposto por meio do *framework* FastAPI, que fornece uma interface leve, eficiente e compatível com os requisitos de comunicação REST. Durante o processo de ingestão, o *backend* em GO enviará os textos ao microserviço Python e armazenará os vetores retornados juntamente aos metadados dos documentos. Isso permitirá que as buscas textuais e semânticas sejam realizadas de forma paralela e posteriormente combinadas por meio de um mecanismo de rankeamento.

Por fim, o armazenamento será feito em um banco de dados PostgreSQL, escolhido por sua confiabilidade, suporte a tipos de dados complexos e compatibilidade com diversas linguagens de programação. Essa combinação de tecnologias visa garantir um sistema robusto, eficiente e adaptável a diferentes contextos de uso.

3.1 Cronograma de Atividades

Figura 10 – Cronograma de atividades.



Fonte: Elaboração própria

Referências

ABREU, L. E. d. L. *Direito como linguagem*. Revista de Antropologia, 2022. Disponível em: <https://revistas.usp.br/ra/article/view/197851>. Acesso em: 14 maio 2025. Citado na página 9.

Agência CNJ de Notícias. *Em 15 anos, Justiça recebeu mais de 250 milhões de processos eletrônicos*. 2024. Portal CNJ. Disponível em: <https://www.cnj.jus.br/em-15-anos-justica-recebeu-mais-de-250-milhoes-de-processos-eletronicos/>. Acesso em: 14 out. 2025. Citado na página 6.

Agência CNJ de Notícias. *Uso de IA no Judiciário cresceu 26% em relação a 2022, aponta pesquisa*. 2024. Portal CNJ. Disponível em: <https://www.cnj.jus.br/uso-de-ia-no-judiciario-cresceu-26-em-relacao-a-2022-aponta-pesquisa/>. Acesso em: 14 out. 2025. Citado na página 6.

ALMEIDA, A. C. M. d.; DIAS, G. Recuperação de informação. In: *Processamento de Linguagem Natural: Conceitos, Técnicas e Aplicações em Português*. 1. ed. Brasileiras em PLN, 2023. cap. 16. Acesso em: 15 out. 2025. Disponível em: <https://brasileiraspln.com/livro-pln/1a-edicao/parte8/cap16/cap16.html>. Citado na página 17.

BLELLOCH, G. E.; MAGGS, B. M. Parallel algorithms. *Communications of the ACM*, 1996. Disponível em: <https://dl.acm.org/doi/pdf/10.1145/234313.234339>. Acesso em: 4 maio 2025. Citado na página 28.

BOYER, J. S. M. R. S. *A Fast String Searching Algorithm*. Communications of the ACM, 1977. Disponível em: <https://dl.acm.org/doi/pdf/10.1145/359842.359859>. Acesso em: 15 maio 2025. Citado 2 vezes nas páginas 25 e 26.

CHALKIDIS MANOS FERGADIOTIS, P. M. I.; ALETRAS, N.; ANDROUTSOPOULOS, I. *LEGAL-BERT: The Muppets straight out of Law School*. 2020. Disponível em: <https://aclanthology.org/2020.findings-emnlp.261.pdf>. Acesso em: 14 maio 2025. Citado 2 vezes nas páginas 23 e 24.

COELHO, S. A. *Introdução a Computação Paralela com o Open MPI*. 2015. Disponível em: <https://www.inf.ufsc.br/~bosco.sobral/ensino/ine5645/DSM/Introducao-a-Computacao-Paralela-com-o-OpenMPI.pdf>. Acesso em: 4 maio 2025. Citado na página 27.

CONTRATRES, F. *Similaridade de vetores*. 2018. Disponível em: <https://medium.com/luizalabs/similaridade-entre-t%C3%ADtulos-de-produtos-com-word2vec-5e26199862f0>. Acesso em: 14 maio 2025. Citado na página 23.

CORMEN, T. H. et al. *Algoritmos: teoria e prática*. 3. ed. Rio de Janeiro: Elsevier, 2013. Citado 2 vezes nas páginas 24 e 26.

COSTA, R. P. D. *Reconhecimento de Entidades Nomeadas em Textos Informais no Domínio Legislativo*. 2023. Disponível em: <https://files.cercomp.ufg.br/weby/up/1289/o/DIssertacao_Final_corrigida.pdf>. Acesso em: 14 maio 2025. Citado na página 24.

DATACAMP. *Os 10 melhores métodos para reduzir os custos do LLM*. 2025. Blog DataCamp. Disponível em: <https://www.datacamp.com/pt/blog/ai-cost-optimization>. Acesso em: 14 out. 2025. Citado na página 6.

DEVLIN MING-WEI CHANG, K. L. J.; TOUTANOVA, K. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. Disponível em: <https://aclanthology.org/N19-1423.pdf>. Acesso em: 16 maio 2025. Citado na página 21.

FLECK, C. *Sistema Operacional Linux em Arquiteturas Multicore*. 2012. Trabalho de Diplomação – Universidade Federal do Rio Grande do Sul. Disponível em: <http://200.20.15.38/~ic2022/wp-content/uploads/2021/07/2012-19.pdf>. Acesso em: 14 out. 2025. Citado na página 7.

GALIL, Z. *On Improving the Worst Case Running Time of the Boyer-Moore String Matching Algorithm*. 1979. Disponível em: <https://dl.acm.org/doi/pdf/10.1145/359146.359148>. Acesso em: 15 maio 2025. Citado na página 26.

GEEKSFORGEEKS. *Removing Stop Words with NLTK in Python*. 2024. Disponível em: <https://www.geeksforgeeks.org/removing-stop-words-nltk-python/>. Acesso em: 12 maio 2025. Citado na página 13.

GUSFIELD, D. *Algorithms on Strings, Trees, and Sequences*. Cambridge University Press, 1997. Disponível em: <https://doc.lagout.org/science/0_Computer%20Science/2_Algorithms/Algorithms%20on%20Strings%2C%20Trees%2C%20and%20Sequences%20%5BGusfield%201997-05-28%5D.pdf>. Acesso em: 15 maio 2025. Citado 2 vezes nas páginas 25 e 26.

HORSPOOL, R. N. *Practical Fast Searching in Strings*. 1980. Disponível em: <https://onlinelibrary.wiley.com/doi/epdf/10.1002/spe.4380100608>. Acesso em: 15 maio 2025. Citado 2 vezes nas páginas 25 e 26.

JURAFSKY, D.; MARTIN, J. H. *Speech and Language Processing*. 2025. Rascunho de 12 de janeiro de 2025. Disponível em: <https://web.stanford.edu/~jurafsky/slp3/>. Acesso em: 4 maio 2025. Citado 5 vezes nas páginas 11, 13, 14, 15 e 23.

KNUTH JAMES HIRAM MORRIS JUNIOR, V. R. P. D. E. *Fast Pattern Matching in Strings*. 1977. Disponível em: <https://www.cs.jhu.edu/~misha/ReadingSeminar/Papers/Knuth77.pdf>. Acesso em: 15 maio 2025. Citado na página 26.

LTDA., E. M. *Michaelis*. 2025. Disponível em: <https://michaelis.uol.com.br/moderno-portugues/busca/portugues-brasileiro/algoritmo/>. Acesso em: 12 maio 2025. Citado na página 24.

MANNING, C. D.; RAGHAVAN, P.; SCHüTZE, H. *Introduction to Information Retrieval*. New York, NY, USA: Cambridge University Press, 2008. Acesso em: 15 out. 2025. Disponível em: <https://nlp.stanford.edu/IR-book/>. Citado 10 vezes nas páginas 11, 13, 14, 15, 16, 17, 18, 19, 22 e 23.

MELLO, M. P. d. *Jürgen Habermas: o Direito como linguagem*. Logeion: Filosofia da Informação, 2020. Disponível em: <https://revista.ibict.br/fiinf/article/view/5148>. Acesso em: 14 maio 2025. Citado 2 vezes nas páginas 9 e 10.

MIKOLOV KAI CHEN, G. C. T.; DEAN, J. *Efficient Estimation of Word Representations in Vector Space*. 2013. Disponível em: <https://arxiv.org/pdf/1301.3781v3.pdf>. Acesso em: 16 maio 2025. Citado 2 vezes nas páginas 19 e 20.

MOTTA, V. S. et al. Uma análise de algoritmos de ranqueamento de texto usando coeficiente de correlação de spearman. In: *Anais do VI Seminário de Tese em Computação (SETC)*. Campo Grande, MS, Brasil: Sociedade Brasileira de Computação (SBC), 2020. Acesso em: 15 out. 2025. Disponível em: <https://sol.sbc.org.br/index.php/setc/article/view/18413>. Citado 2 vezes nas páginas 24 e 25.

NUNES, E.; MENDONçA, I.; RALHA, C. Análise comparativa do bert e chatgpt no reconhecimento de entidades nomeadas do domínio jurídico. In: *Anais da XXII Escola Regional de Computação Aplicada a Saúde (ERCAS 2024)*. Porto Alegre, RS, Brasil: SBC, 2024. Acesso em: 14 out. 2025. Disponível em: <https://sol.sbc.org.br/index.php/reic/article/view/3903>. Citado na página 7.

OLIVEIRA, H. d. S.; RODRIGUES, D. C.; APPEL, A. P. Preprocessing applied to legal text mining: analysis and evaluation of the main techniques used. In: *Anais do Encontro Nacional de Inteligência Artificial e Computacional (ENIAC)*. [s.n.], 2022. Disponível em: <https://sol.sbc.org.br/index.php/eniac/article/view/25760/25576>. Acesso em: 4 maio 2025. Citado na página 11.

PAULA, L. C. M. d. *Paralelização de Algoritmos APS e Firefly para Seleção de Variáveis em Problemas de Calibração Multivariada*. 2019. Dissertação de Mestrado. Disponível em: <https://ww2.inf.ufg.br/ppgcc/sites/www.inf.ufg.br.mestrado/files/uploads/Dissertacoes/Disserta%C3%A7%C3%A3o%20Lauro%20C%C3%A1ssio.pdf>. Acesso em: 4 maio 2025. Citado 2 vezes nas páginas 27 e 28.

PAULA, L. F. de et al. Pln e segurança jurídica: Identificação de divergências jurisprudenciais com processamento de linguagem natural. In: *Anais do XVIII Simpósio de Tecnologia da Informação e de Sistemas de Informação (STISI)*. SBC, 2024. Acesso em: 14 out. 2025. Disponível em: <https://sol.sbc.org.br/index.php/stil/article/view/31161>. Citado na página 6.

PENNINGTON RICHARD SOCHER, C. D. M. J. *GloVe: Global Vectors for Word Representation*. 2014. Disponível em: <https://aclanthology.org/D14-1162.pdf>. Acesso em: 16 maio 2025. Citado 2 vezes nas páginas 20 e 21.

PETERS MARK NEUMANN, M. I. M. E. et al. *Deep Contextualized Word Representations*. 2018. Disponível em: <https://aclanthology.org/N18-1202.pdf>. Acesso em: 16 maio 2025. Citado na página 21.

PINHEIRO, G. M. *Direito e linguagem: um estudo sobre a influência da filosofia da linguagem na teoria do direito*. Revista de Teorias da Justiça, da Decisão e da Argumentação Jurídica, 2023. Disponível em: <https://indexlaw.org/index.php/revistateoriasjustica/article/view/9595>. Acesso em: 14 maio 2025. Citado na página 9.

RAHUL. *TF-IDF: Understanding Term Frequency-Inverse Document Frequency in NLP*. Zilliz, 2023. Disponível em: <https://zilliz.com/learn/tf-idf-understanding-term-frequency-inverse-document-frequency-in-nlp>. Acesso em: 14 maio 2025. Citado na página 15.

REIMERS, I. G. N. *Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks*. 2019. Disponível em: <https://aclanthology.org/D19-1410.pdf>. Acesso em: 13 maio 2025. Citado 5 vezes nas páginas 19, 20, 21, 23 e 24.

RODRIGUES, R. G. *Aplicação de RAG para identificação de atos judiciais similares na Justiça Eleitoral*. 2024. Trabalho de Conclusão de Curso (Sistemas de Informação) - Universidade de São Paulo (USP). Acesso em: 15 out. 2025. Disponível em: <https://bdta.abcd.usp.br/directbitstream/47d820ca-56f9-4ef0-b0c7-95df6e1b012e/Ramon_Gouveia_Rodrigues.pdf>. Citado na página 17.

SANTOS, K. H. R. d. *Classificação de textos de petições jurídicas com base em técnicas de Processamento de Línguas Naturais*. Dissertação (Mestrado) — Universidade de São Paulo, São Paulo, 2023. Acesso em: 14 out. 2025. Disponível em: <https://bdta.abcd.usp.br/directbitstream/10cbe4f4-36c7-4c97-bd82-1477ec3348b4/Kelsen%20Henrique%20Rolim%20dos%20Santos.pdf>. Citado na página 6.

SAP. *Entendendo o processamento de linguagem natural: um guia*. s.d. Disponível em: <https://www.sap.com/brazil/resources/what-is-natural-language-processing>. Acesso em: 14 out. 2025. Citado na página 7.

SEZERER, S. T. E. *A Survey On Neural Word Embeddings*. 2021. Disponível em: <https://arxiv.org/pdf/2110.01804>. Acesso em: 16 maio 2025. Citado 2 vezes nas páginas 19 e 20.

SIEGEL, S.; CASTELLAN N. JOHN, J. *Nonparametric Statistics for the Behavioral Sciences*. 2. ed. New York: McGraw-Hill, 1988. Citado 2 vezes nas páginas 24 e 25.

SOUZA, L. N. S. d. *Direito e linguagem: como a linguagem codifica o direito?* Universidade Federal de Santa Catarina, 2023. Disponível em: <https://repositorio.ufsc.br/handle/123456789/258418>. Acesso em: 14 maio 2025. Citado 2 vezes nas páginas 9 e 10.

TURING. *Stemming vs Lemmatization in Python: What's the Difference?* 2023. Artigo publicado no Turing Knowledge Base. Disponível em: <https://www.turing.com/kb/stemming-vs-lemmatization-in-python>. Acesso em: 6 maio 2025. Citado 2 vezes nas páginas 13 e 14.

ZAMPIERI, M. et al. *LegalNLP – Natural Language Processing methods for the Brazilian Legal Language*. 2021. Artigo publicado no arXiv:2110.15709. Disponível em: <https://arxiv.org/abs/2110.15709>. Acesso em: 6 maio 2025. Citado 5 vezes nas páginas 9, 10, 12, 13 e 14.

ZIVIANI, N. *Projeto de Algoritmos Com Implementações em Pascal e C*. 2004. Livro eletrônico. Disponível em: <https://d1wqtxts1xzle7.cloudfront.net/52574456/ebook_Projetos_de_Algoritmos_Com_Implementacoes_em_Pascal_e_C_Nivio_Ziviani__4ed-libre.pdf>. Acesso em: 4 maio 2025. Citado na página 27.