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: Afonso Lelis

São Paulo - Brasil
2025

Dedicamos este trabalho aos nossos pais, que sempre foram nosso alicerce, oferecendo o apoio incondicional e o incentivo necessário para que pudéssemos trilhar este caminho de aprendizado. A todos que acreditaram no nosso potencial e nos motivaram a nunca desistir dos nossos sonhos.

Agradecimentos

Primeiramente, agradecemos aos nossos pais, nossa eterna gratidão pelo amor, paciência e apoio incondicional. Vocês são a base de todas as nossas conquistas.

Ao nosso orientador, Professor Afonso Lelis, e ao Centro Universitário Senac, agradecemos pela orientação segura, pela estrutura oferecida e pelos conhecimentos compartilhados, que foram essenciais para o desenvolvimento deste trabalho e para a nossa formação profissional.

Por fim, aos amigos e colegas de curso, obrigado pelo companheirismo e pela troca de experiências que tornaram essa caminhada mais leve.

"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)"

Resumo

A digitalização massiva do setor jurídico brasileiro impõe desafios críticos de recuperação de informação. Enquanto modelos de Inteligência Artificial baseados em *embeddings* oferecem alta precisão semântica, seu custo computacional é elevado. Este trabalho investiga o *trade-off* pragmático entre complexidade semântica e eficiência estatística. Para isso, foi desenvolvido um sistema comparativo integralmente em linguagem Go, implementando os algoritmos TF-IDF e Okapi BM25, confrontados com um modelo *transformer* (Sentence-BERT). Os testes, realizados sobre um *corpus* legislativo de aproximadamente 5.000 documentos, revelaram que a busca estatística é significativamente mais eficiente: o uso de Unigramas exigiu apenas 143 MB de memória RAM, contra 4,7 GB na configuração de Trigramas. Em termos de velocidade, o algoritmo BM25 mostrou-se até 10 vezes mais rápido que o TF-IDF em cenários complexos. A análise qualitativa via Coeficiente de Spearman indicou uma correlação baixa (entre 0,02 e 0,06) entre os *rankings* estatísticos e semânticos, sugerindo que as abordagens capturam aspectos distintos da relevância. Conclui-se que, para o domínio jurídico, métodos estatísticos oferecem um custo-benefício superior para filtragem inicial, sendo a arquitetura híbrida a solução ideal.

Palavras-chave: Processamento de Linguagem Natural. Recuperação de Informação. Busca Jurídica. TF-IDF. BM25. Embeddings.

Abstract

The massive digitization of the Brazilian legal sector imposes critical information retrieval challenges. While AI models based on embeddings offer high semantic precision, their computational cost is high. This work investigates the pragmatic trade-off between semantic complexity and statistical efficiency. To this end, a comparative system was developed entirely in Go, implementing TF-IDF and Okapi BM25 algorithms, confronted with a transformer model (Sentence-BERT). Tests performed on a legislative corpus of approximately 5,000 documents revealed that statistical search is significantly more efficient: using Unigrams required only 143 MB of RAM, versus 4.7 GB in the Trigram configuration. In terms of speed, the BM25 algorithm proved to be up to 10 times faster than TF-IDF in complex scenarios. Qualitative analysis via Spearman's Coefficient indicated a low correlation (between 0.02 and 0.06) between statistical and semantic rankings, suggesting that the approaches capture distinct aspects of relevance. It is concluded that, for the legal domain, statistical methods offer superior cost-benefit for initial filtering, with hybrid architecture being the ideal solution.

Keywords: Natural Language Processing. Information Retrieval. Legal Search. TF-IDF. BM25. Embeddings.

Lista de ilustrações

Lista de abreviaturas e siglas

API	Application Programming Interface
BERT	Bidirectional Encoder Representations from Transformers
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
ORM	Object-Relational Mapping
GORM	Golang ORM
I/O	Input/Output
HTML	HyperText Markup Language
PL	Proposta de lei
PEC	Proposta de Emenda à Constituição

SSD	Solid-State Drive
HDD	Hard Disk Drive
RAM	Random Access Memory
NVMe	Non-Volatile Memory Express
SQL	Structured Query Language

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 de forma contundente na digitalização massiva de processos, leis, jurisprudências e pareceres. O Judiciário brasileiro, por exemplo, vivenciou uma explosão documental ao receber 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 inegável em termos de transparência e preservação, por outro, ela impõe um desafio monumental: a simples capacidade de armazenamento tornou-se insuficiente. O problema agora desloca-se para a navegabilidade e a extração de valor desse oceano de textos complexos e interconectados, criando uma demanda urgente não apenas por memória digital, mas por capacidade de processamento inteligente.

Para responder a essa necessidade de inteligência interpretativa, as técnicas de Processamento de Linguagem Natural (NLP), especialmente aquelas baseadas em Inteligência Artificial e aprendizado de máquina, emergiram como a fronteira tecnológica mais promissora. Diferentemente dos sistemas antigos de busca exata, os modelos de linguagem avançados — notadamente aqueles baseados na arquitetura *Transformer*, como o BERT e o LEGAL-BERT — prometem "ler"os documentos, compreendendo nuances semânticas e contextuais que escapam à busca por palavras-chave. Ferramentas que utilizam arquiteturas de *transformers* e representações vetoriais (*embeddings*) representam o atual estado da arte, buscando emular uma compreensão quase humana dos textos legais para resolver tarefas complexas, como a identificação de similaridade entre decisões e divergências jurisprudenciais (PAULA et al., 2024).

No entanto, o entusiasmo com a sofisticação da Inteligência Artificial muitas vezes ofusca uma barreira prática significativa: o custo computacional. A adoção dessas tecnologias de ponta não é gratuita; ela exige um esforço financeiro, técnico e energético considerável. O treinamento e, principalmente, a inferência de grandes modelos de linguagem demandam hardware especializado (GPUs) e infraestrutura robusta, o que pode tornar a solução inviável para muitas instituições. Conforme o uso de IA se expande, torna-se crítico não apenas buscar o modelo mais "inteligente", mas também aplicar métodos para otimizar e reduzir os custos de operação desses sistemas, garantindo sua sustentabilidade (DATACAMP, 2025).

Essa tensão entre a performance semântica e o custo computacional nos leva diretamente ao cerne da investigação proposta neste trabalho. Diante de um cenário onde recursos são finitos, é imperativo questionar até que ponto a complexidade dos modelos de *machine learning* é estritamente indispensável para a tarefa de busca em documentos jurídicos. Surge, então, uma dúvida metodológica relevante: será que métodos estatísticos e algoritmos de busca mais clássicos, se bem parametrizados e aplicados, não poderiam alcançar resultados comparáveis aos da IA moderna, mas consumindo apenas uma fração dos recursos?

Este estudo se propõe, portanto, a explorar essa fronteira de eficiência. A literatura

já aponta indícios nessa direção, como observado por Santos (2023), que demonstra que modelos distribucionais complexos como o BERT não apresentam, necessariamente, um desempenho superior em todas as tarefas de classificação jurídica quando comparados a abordagens mais simples. O objetivo deste trabalho não é desenvolver uma nova ferramenta, mas sim realizar uma análise crítica e comparativa rigorosa, buscando compreender e quantificar o verdadeiro *trade-off* entre a precisão semântica oferecida pela IA e a eficiência computacional dos métodos estatísticos no domínio específico do Direito.

1.2 Justificativa

A busca por maior precisão impulsionou o avanço da tecnologia jurídica e estimulou uma dependência crescente de modelos de Inteligência Artificial. Esse movimento ganhou força no Brasil, onde o uso dessas soluções no Judiciário aumentou 26% entre 2022 e 2023 (Agência CNJ de Notícias, 2024b). A ideia que sustenta esse crescimento é simples. Quanto mais sofisticado o modelo, melhores seriam os resultados. Só que essa expectativa costuma vir acompanhada de um custo elevado. O treinamento de grandes modelos depende de processamento intenso e de hardware especializado, o que envolve investimentos altos em infraestrutura e manutenção. Em muitos casos, o peso financeiro impede que instituições menores entrem nessa corrida tecnológica, mesmo quando há interesse.

Ao mesmo tempo, a evolução do hardware abriu caminhos que nem sempre são lembrados. O avanço das arquiteturas *multicore* não transformou apenas os modelos de IA. Ele também ampliou o alcance de algoritmos clássicos que antes operavam de forma limitada. Hoje esses métodos podem explorar vários núcleos de processamento e alcançar um desempenho muito superior ao que entregavam no passado (FLECK, 2012). Técnicas como TF-IDF, assim como algoritmos de busca de padrões como o Boyer-Moore, não exigem estruturas complexas e ainda assim se beneficiam de forma notável da paralelização. Mesmo com esse potencial, costumam ser vistos como alternativas inferiores simplesmente por não pertencerem ao universo dos modelos semânticos mais recentes.

Essa percepção costuma levar a decisões pouco equilibradas. Por isso, a justificativa deste trabalho parte da necessidade de um olhar mais pragmático. Nem sempre o ganho marginal oferecido por um modelo semântico avançado compensa seu custo total de operação. Em ambientes com linguagem estável e previsível, como ocorre em grande parte dos documentos jurídicos, soluções mais simples podem alcançar resultados muito satisfatórios (SAP, s.d.). Em várias atividades do dia a dia, responder rápido e gastar menos pode ser mais importante do que captar nuances raras de ambiguidade.

Dentro desse cenário, esta pesquisa busca oferecer uma análise quantitativa do equilíbrio entre custo e benefício. A proposta é comparar diferentes abordagens em termos de desempenho, tempo de resposta e uso de recursos. Trabalhos que analisam modelos como BERT e ChatGPT já fazem esse tipo de avaliação usando métricas como acurácia, precisão e F1-score (NUNES; MENDONçA; RALHA, 2024). Seguindo essa linha, pretendemos produzir dados concretos que ajudem a escolher a tecnologia mais adequada para cada necessidade do ecossistema jurídico. A intenção é mover a discussão para um campo mais objetivo, onde a escolha não dependa apenas do que é mais moderno, mas do que realmente funciona melhor dentro de cada contexto.

1.3 Objetivos

1.3.1 Objetivo Geral

Avaliar comparativamente o desempenho computacional e a relevância semântica de métodos estatísticos, especificamente o **TF-IDF e o BM25**, frente a modelos baseados em *embeddings* como o **Sentence-BERT**, visando analisar o *trade-off* entre eficiência e precisão na recuperação de informação em documentos legislativos.

1.3.2 Objetivos Específicos

1. Construir um *corpus* textual robusto a partir da coleta automatizada e higienização de documentos legislativos, com foco em Propostas de Lei e Propostas de Emenda à Constituição obtidas do portal da Câmara dos Deputados.
2. Implementar e parametrizar os algoritmos de busca estatística **TF-IDF e BM25**, bem como a abordagem semântica via **Sentence-BERT**, estruturando estratégias de indexação com diferentes configurações de N-gramas.
3. Definir um protocolo experimental para mensurar o consumo de recursos, incluindo tempo de CPU e uso de memória RAM, bem como a latência de indexação e busca em diferentes cenários de carga.
4. Aplicar o Coeficiente de Correlação de Spearman para quantificar o grau de concordância entre os *rankings* gerados pelos métodos estatísticos e o gabarito semântico.
5. Analisar quantitativamente os resultados para determinar em quais cenários o custo computacional dos modelos de IA se justifica frente à eficiência dos métodos clássicos.

1.4 Metodologia

Esta pesquisa segue um procedimento metodológico estruturado e replicável, detalhado nas etapas a seguir.

1.4.1 Materiais e Ferramentas

Para a execução dos experimentos e o processamento do volume de dados, foram utilizados os seguintes recursos:

- **Hardware:** MacBook Pro 2020, Chip M1, 16 GB de RAM (4266 MHz).
- **Dados:** Base de dados pública da Câmara dos Deputados.
- **Software:** Linguagem Go (Golang), versão 1.23.2.

1.4.2 Configurações e Parâmetros

Os modelos e algoritmos foram configurados com os seguintes parâmetros:

- O modelo BERT selecionado foi o `paraphrase-multilingual-MiniLM-L12-v2`.

- Para o algoritmo BM25, os hiperparâmetros foram fixados em $k_1 = 1.5$ e $b = 0.75$.
- Todos os resultados de *embeddings* e os cálculos de similaridade de cosseno foram normalizados.
- Na geração de n-gramas, definiu-se um limite de até 2 saltos (*jumps*) para trigramas e 4 saltos para bigramas.

1.4.3 Tratamento do Corpus

1. **Definição:** O corpus é composto por 5 mil páginas de documentos em PDF, abrangendo Projetos de Lei (PL) e Propostas de Emenda à Constituição (PEC).
2. **Extração:** O conteúdo dos PDFs foi extraído e convertido para texto puro.
3. **Padronização:** Todo o texto foi convertido para letras minúsculas.
4. **Limpeza:** O corpus passou por um processo de limpeza seguindo esta ordem:

   Remoção de URIs (conforme RFC 3986) e protocolos HTTP/TLS.

   Remoção de numerais romanos, inteiros e decimais.

   Substituição de travessões e *underlines* por espaços.

   Eliminação de pontuações e caracteres especiais restantes.
5. **Finalização:** Realizou-se a normalização de caracteres e a remoção de *stop-words*.

1.4.4 Geração das Embeddings

1. Foram elaborados 50 *prompts* com tamanhos de 10, 20 e 40 palavras, simulando buscas reais de um operador do Direito.
2. Cada *prompt* foi vetorizado via BERT em vetores de 384 pontos flutuantes. O tempo de cada operação foi registrado para extração de médias.
3. O mesmo processo de vetorização foi aplicado a todos os documentos do corpus.
4. A comparação entre as *embeddings* dos documentos e dos *prompts* foi realizada por meio da similaridade de cosseno.
5. Os documentos foram ordenados por relevância para cada *prompt*, gerando um ranking final.

1.4.5 Geração de Vetores TF-IDF e BM25

1. Os ambientes de teste foram isolados por tamanho de n-grama (uni, bi e trigramas), com o banco de dados sendo zerado a cada rodada para evitar interferências.
2. Foram gerados identificadores únicos para documentos, palavras e n-gramas.
3. Os n-gramas foram formados pela combinação dos IDs das palavras, saltos realizados, documento de origem e frequência.

4. Implementou-se um cache em duas camadas (*Map[Chave do n-grama]* → *Map[ID do documento]* → *N-grama*) para otimização de performance.

5. Após o processamento do corpus e dos *prompts*, os vetores TF-IDF e BM25 foram gerados, registrando-se os tempos de execução.

6. A similaridade de cosseno foi aplicada para ordenar os documentos conforme sua relevância em relação aos *prompts*.

1.4.6 Análise dos Dados

A análise dos resultados será realizada através de duas abordagens complementares, permitindo uma visão detalhada sobre a eficácia de cada método.

1.4.6.1 Análise Qualitativa e Comparação de Rankings

Para avaliar a proximidade entre os métodos (BERT, TF-IDF e BM25), utilizaremos o **Coeficiente de Correlação de Spearman**. Esta métrica é ideal para este estudo pois ela não olha apenas para os valores brutos de similaridade, mas sim para a ordenação (*ranking*) dos documentos. Ao mensurar a diferença de posição dos documentos entre as listas geradas, o coeficiente de Spearman permite identificar o grau de convergência entre os algoritmos. Dessa forma, conseguimos medir qualitativamente se um método baseado em semântica (BERT) entrega resultados hierarquicamente próximos aos métodos baseados em frequência de termos (BM25/TF-IDF).

1.4.6.2 Análise Quantitativa de Desempenho

A vertente quantitativa focará na eficiência computacional. Utilizaremos os registros de tempo coletados durante as etapas de vetorização e comparação para analisar:

- O tempo médio de processamento por documento e por *prompt*.
- A soma total de tempo necessária para processar o corpus completo em cada abordagem.
- A escalabilidade dos métodos em relação ao tamanho dos textos (10, 20 e 40 palavras).

O cruzamento desses dados permitirá identificar o custo-benefício de cada modelo, determinando qual técnica oferece a melhor relação entre precisão na busca e latência de resposta.

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 de difícil processamento. No universo jurídico, esse fenômeno é ainda mais crítico, visto que contratos, decisões e legislações são produtos essencialmente linguísticos. O Direito, enquanto sistema normativo, é construído e operado exclusivamente por meio da linguagem, o que torna a precisão interpretativa um requisito fundamental (ABREU, 2022; MELLO, 2020).

Para lidar com esse volume exponencial de documentos, que excede as limitações cognitivas humanas, o uso de ferramentas computacionais deixou de ser opcional para se tornar uma necessidade. Nesse cenário, as técnicas de Processamento de Linguagem Natural (NLP) surgem não apenas como uma ferramenta de automação, mas como a abordagem consolidada para enfrentar o desafio de interpretar grandes massas de dados textuais (ZAMPIERI et al., 2021; SOUZA, 2023).

Diferentemente de sistemas rígidos baseados em regras, o NLP permite que as máquinas compreendam e operem diretamente sobre a base textual do Direito. Essas técnicas viabilizam desde a extração de informações relevantes e identificação de padrões até a sumarização de documentos e suporte à tomada de decisão, incorporando uma camada de "inteligência"ao processamento bruto de dados (PINHEIRO, 2023; MELLO, 2020).

Essa evolução tecnológica já se traduz em aplicações práticas robustas no ambiente jurídico, como a análise preditiva de jurisprudência e a busca semântica em bases normativas. No contexto brasileiro, estudos como os de Zampieri et al. (2021) evidenciam que essas tecnologias já estão consolidadas, estabelecendo um novo padrão de referência para a recuperação e análise de informação no setor.

2.1.1 Características dos textos jurídicos

A aplicação de técnicas de recuperação de informação no domínio jurídico envolve desafios particulares, uma vez que os textos legais apresentam propriedades distintas de outros gêneros textuais. Em primeiro lugar, são documentos altamente especializados, redigidos com vocabulário técnico e linguagem formal. Termos como *vossa excelência*, *ex positis* ou *ad nutum* desempenham papel semântico central (ABREU, 2022; SOUZA, 2023). Para sistemas de busca, essa especificidade lexical gera efeitos distintos dependendo da abordagem: favorece métodos baseados em palavras-chave (devido à raridade dos termos), mas pode confundir modelos semânticos genéricos que não foram treinados nesse vocabulário específico.

Além disso, esses textos apresentam um alto grau de ambiguidade controlada. Palavras como *poderá*, *deverá* ou *caberá* têm significados normativos específicos que impactam diretamente a interpretação das normas. Essa polissemia representa um desafio de desambiguação: métodos estatísticos tendem a tratar essas palavras apenas como *tokens* distintos, ignorando sua carga normativa, enquanto modelos semânticos precisam de um

ajuste fino robusto para diferenciar o "sentido jurídico"do uso coloquial (ZAMPIERI et al., 2021; PINHEIRO, 2023).

Outro aspecto crítico, e talvez o mais desafiador para a recuperação de informação, é a intertextualidade jurídica. Leis, contratos e sentenças não existem no vácuo; eles referenciam constantemente outros documentos. Isso cria um cenário onde a relevância de um documento muitas vezes depende de uma referência externa exata, e não apenas do seu conteúdo semântico interno. Esse fenômeno desafia a lógica dos *embeddings*, que buscam similaridade de significado e podem perder a precisão da referência exata, e também impõe dificuldades aos métodos estatísticos, que tratam citações complexas apenas como sequências de caracteres (ZAMPIERI et al., 2021).

Também é importante considerar a estrutura argumentativa dos textos. Documentos como sentenças possuem uma lógica discursiva específica. A informação mais relevante para uma busca nem sempre está distribuída uniformemente pelo texto. Isso afeta diretamente a performance dos algoritmos: o TF-IDF pode supervalorizar a repetição de termos na fundamentação, enquanto um modelo de *embedding* que faz a média vetorial do documento inteiro pode diluir o "sinal"da decisão final contida na conclusão (MELLO, 2020).

2.1.2 Desafios de Processamento e Representação em Português

Ao aplicar técnicas de NLP à língua portuguesa, especialmente no domínio jurídico, surgem obstáculos que permeiam desde a análise morfossintática até a representação semântica vetorial. Diferentemente do inglês, o português brasileiro apresenta uma morfologia altamente flexiva, com ampla variação de formas verbais, nominais e pronominais, o que eleva a complexidade de tarefas fundamentais como a tokenização e a lematização (ZAMPIERI et al., 2021).

No contexto jurídico, essa complexidade é intensificada pelo uso recorrente de locuções e expressões idiomáticas. Construções como *sem prejuízo do disposto*, *salvo disposição em contrário* ou *nos termos da legislação aplicável* funcionam como unidades semânticas coesas. Analisá-las isoladamente, palavra por palavra, pode fragmentar o sentido original, exigindo abordagens de pré-processamento capazes de identificar e preservar o significado completo dessas expressões (SOUZA, 2023).

Essa especificidade linguística impõe desafios práticos imediatos na etapa de tokenização. O tratamento da pontuação, por exemplo, exige regras de exceção: elementos como números de processo (ex: 1234567-89.2023.8.26.0100) devem ser preservados como um único *token*, dada sua importância identificadora, enquanto a pontuação gramatical comum deve ser removida (ZAMPIERI et al., 2021). Da mesma forma, o uso de hífens em palavras compostas (ex: *auto-aplicável*) e a presença de entidades nomeadas complexas (ex: *Ministério Público*, *Poder Judiciário*) demandam estratégias de segmentação que evitem a perda de informação semântica, um problema bem documentado na literatura de NLP aplicada ao direito brasileiro (ZAMPIERI et al., 2021).

Além da segmentação e limpeza, há o desafio da representação vetorial (*embeddings*). Para que algoritmos possam realizar tarefas como busca jurisprudencial ou classificação, os textos precisam ser convertidos em vetores numéricos. No entanto, modelos de *embeddings* genéricos, treinados com textos da web, frequentemente falham em capturar as nuances da linguagem técnica e formal do Direito (CHALKIDIS MANOS FERGADIOTIS; ALETRAS; ANDROUTSOPOULOS, 2020).

Modelos especializados, como a família LEGAL-BERT (CHALKIDIS MANOS FERGADIOTIS; ALETRAS; ANDROUTSOPOULOS, 2020), que são pré-treinados ou ajustados (*fine-tuned*) especificamente em corpora jurídicos, mostram-se mais eficazes na captação desse vocabulário especializado e das estruturas discursivas recorrentes. Contudo, o uso desses modelos introduz um novo obstáculo: o alto custo computacional de treinamento e inferência. Isso levanta a questão central do *trade-off* entre a precisão semântica oferecida por modelos de linguagem pesados e a eficiência de métodos mais simples, cerne da investigação deste trabalho (REIMERS, 2019; COSTA, 2023).

2.1.3 Modelos de Linguagem para o Domínio Jurídico

Embora bibliotecas de NLP de propósito geral, como *spaCy* ou *NLTK*, ofereçam suporte para a língua portuguesa, seus modelos genéricos, treinados em textos da web ou literatura, falham em capturar as nuances terminológicas e o contexto semântico específico do universo jurídico. A palavra "competência", por exemplo, tem um significado técnico e distinto no Direito (atribuição de um órgão para julgar) que um modelo genérico não consegue priorizar.

Para suprir essa lacuna, a comunidade acadêmica tem se concentrado no desenvolvimento de modelos de linguagem especializados. Um trabalho central nesse contexto para o português brasileiro é o Zampieri et al. (2021) (LegalNLP). Os autores demonstram que modelos como Word2Vec e BERT, quando pré-treinados ou ajustados (*fine-tuning*) em um vasto e específico corpus de documentos jurídicos nacionais, superam significativamente o desempenho de modelos genéricos em tarefas-chave de NLP.

Essa abordagem de especialização é uma tendência global. O trabalho de Chalkidis Manos Fergadiotis, Aletras e Androutsopoulos (2020), por exemplo, introduziu o LEGAL-BERT, um modelo da família BERT treinado inteiramente em textos legais. A pesquisa comprovou que essa especialização de domínio permite uma captação muito mais precisa do vocabulário técnico e das estruturas discursivas do Direito do que o modelo BERT original.

Portanto, a literatura estabelece que o estado da arte para a busca semântica de alta precisão no domínio jurídico envolve o uso de modelos de linguagem contextuais, pesados e de domínio específico. É justamente essa abordagem, que representa o polo de maior complexidade e custo computacional, que este trabalho utilizará como referência para a comparação com os métodos estatísticos tradicionais.

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

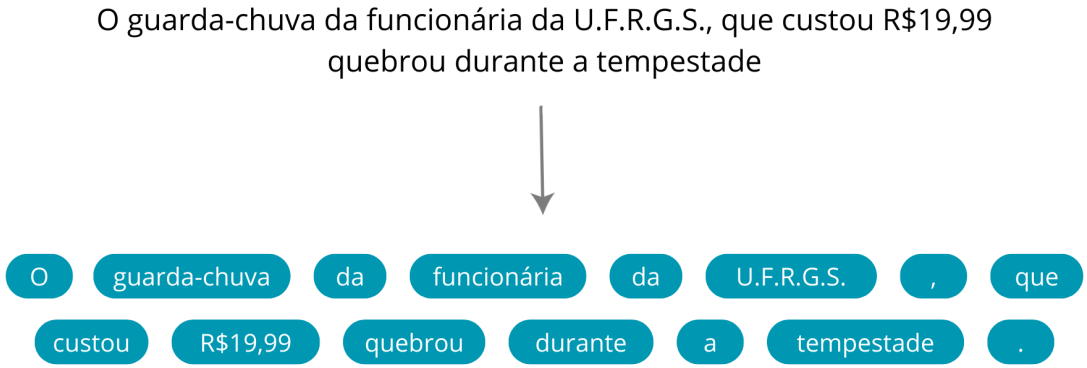
pela definição do vocabulário de termos que será utilizado na indexação e na construção de representações computacionais. Um bom pré-processamento reduz o tamanho do vocabulário, melhora o desempenho e aumenta a precisão das buscas (MANNING; RAGHAVAN; SCHüTZE, 2008). Essa etapa funciona, portanto, como 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*.



Fonte: Elaboração própria.

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*

Texto com Stopwords	Sem Stopwords
Aprender programação é uma habilidade importante para quem trabalha com tecnologia.	Aprender, Programação, Habilidade, Importante, Trabalha, Tecnologia
Será que estudar todos os dias realmente faz diferença?	Estudar, Dias, Realmente, Diferença
Eu gosto de ler livros, mas às vezes não tenho tempo.	Gosto, Ler, Livros, Vezes, Tempo

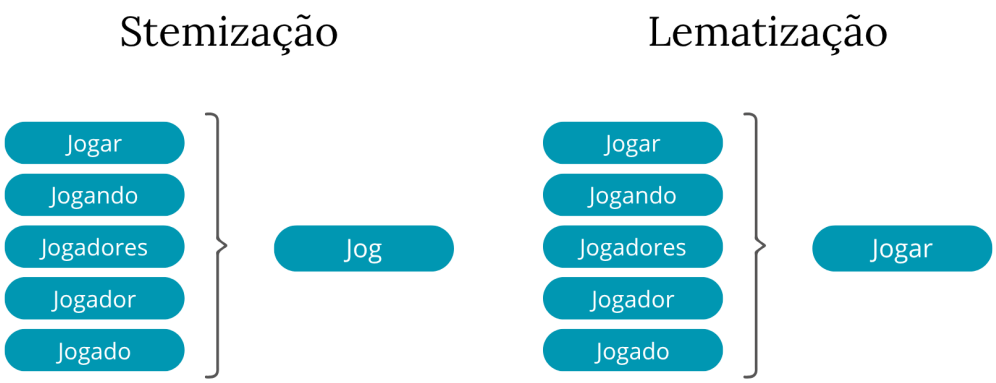
Fonte: Elaboração própria.

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: Elaboração própria, inspirada em (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, que é 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
	documento	e	este	o	primeiro	segundo	terceiro	é
"Este é o primeiro documento"	0,30217	0	0,20000	0,20000	0,30217	0	0	0,20000
"Este documento é o segundo documento"	0,50361	0	0,16667	0,16667	0	0,31938	0	0,16667
"E este é o terceiro"	0	0,38326	0,20000	0,20000	0	0	0,38326	0,20000
"Este é o primeiro?"	0	0	0,25000	0,25000	0,37771	0	0	0,25000

Fonte: Elaboração própria, inspirada em (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).

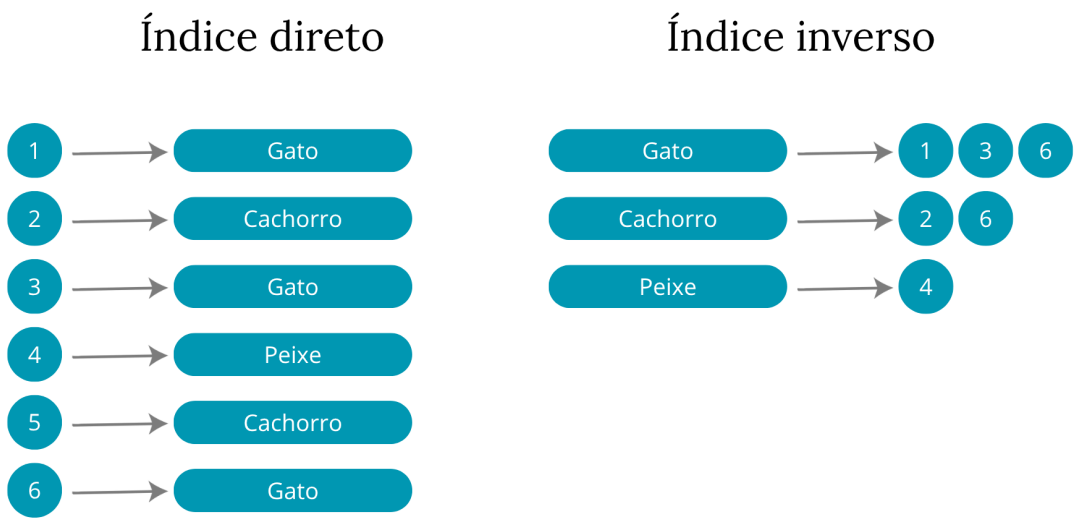
2.3.3 Índice inverso

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 A Evolução dos Modelos de *Embeddings*

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

2.3.4.2.1 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.

2.3.4.2.2 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.

2.3.4.3 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.

2.3.4.3.1 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).

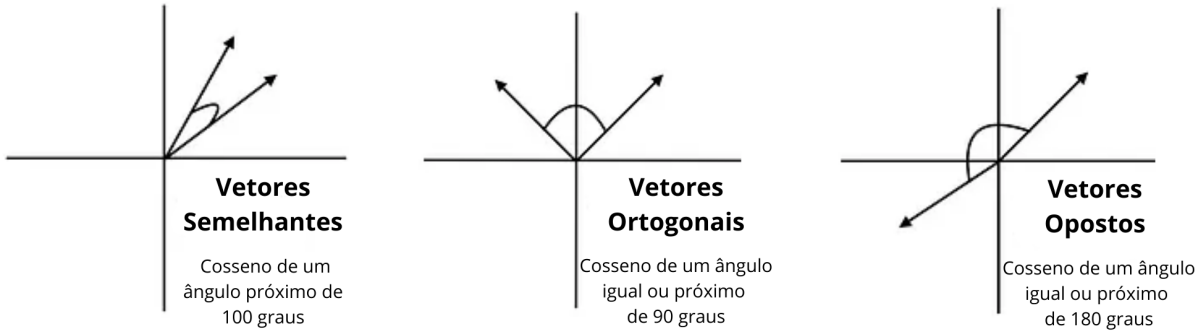
A fórmula da similaridade de cosseno entre dois vetores $\vec{A}$ e $\vec{B}$ é dada por:

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

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.

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: Elaboração própria, inspirada em (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).

2.4 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 et al. (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).

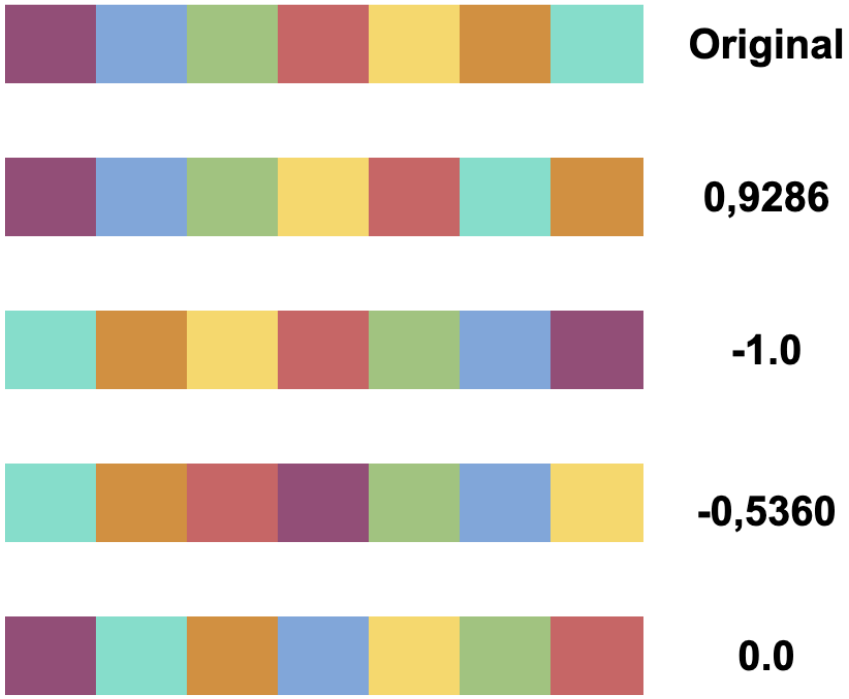
Na prática, o coeficiente de *Spearman* é simplesmente definido como o **coeficiente de correlação de *Pearson* calculado sobre os postos das variáveis**. O processo de cálculo envolve transformar os *scores* de cada lista em uma sequência de postos (1º, 2º, 3º, ...). Caso ocorram empates (por exemplo, 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.

Figura 7 – Demonstração do coeficiente de *Spearman* aplicado à sequência de cores



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 Figura 8.

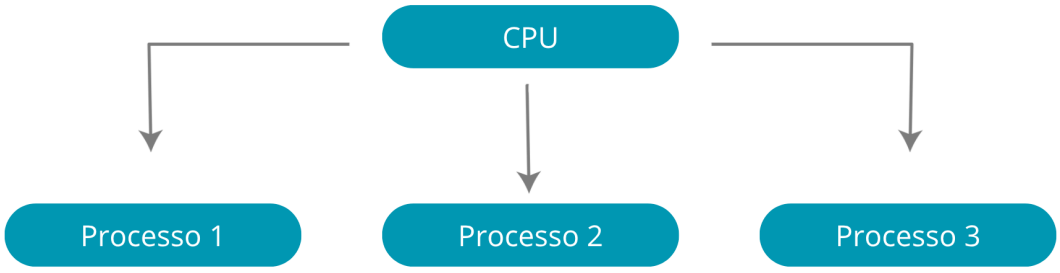
Figura 8 – Algoritmos sequenciais

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

2.6.1 A Consolidação do BM25

O trabalho fundamental de Robertson et al. (1995) na conferência TREC-3 estabeleceu o Okapi BM25 como o estado da arte em modelos probabilísticos. Ao comparar o BM25 com o tradicional TF-IDF em um corpus diversificado, os autores demonstraram que a normalização pelo tamanho do documento e a saturação da frequência de termos corrigiam distorções de relevância em textos longos.

2.6.2 Eficiência em Representações Vetoriais

Em "Efficient Estimation of Word Representations in Vector Space", Mikolov et al. (2013) mudaram o paradigma da busca semântica ao propor o Word2Vec. O estudo focou no *trade-off* computacional, comparando as arquiteturas CBOW (mais rápida) e Skip-Gram (mais precisa semanticamente). Os autores concluíram que é possível treinar modelos de alta qualidade em corpora massivos com custos de CPU aceitáveis, desde que a arquitetura seja simplificada (removendo camadas ocultas não lineares).

2.6.3 Comparação no Domínio Jurídico Brasileiro

Mais recentemente, no contexto nacional, Santos (2023) investigou a classificação de petições jurídicas comparando modelos clássicos, como Naive Bayes e SVM com TF-IDF, contra modelos baseados em *Transformers* (BERT). Os resultados mostraram que, embora o BERT ofereça vantagens em tarefas complexas, modelos mais simples baseados em frequência de termos ainda apresentam desempenho competitivo com um custo computacional drasticamente menor.

3 Desenvolvimento

O desenvolvimento da solução foi guiado por requisitos não funcionais de alto desempenho e eficiência no uso de recursos, essenciais para o processamento de grandes volumes de dados jurídicos. A arquitetura do sistema foi projetada de forma monolitica e autocontida, visando eliminar a latência de rede típica de microsserviços e maximizar o aproveitamento da memória RAM.

Para materializar essa arquitetura, optou-se pela utilização da linguagem Go. Essa escolha foi estritamente técnica, motivada pela capacidade da linguagem de gerar binários compilados nativos e pelo seu modelo eficiente de concorrência. No entanto, os conceitos e algoritmos aqui apresentados são agnósticos à tecnologia e poderiam ser reproduzidos em outras linguagens de sistema.

As seções seguintes descrevem o ciclo de vida da informação dentro do sistema, desde a estruturação do corpus até a execução dos algoritmos de similaridade.

3.1 Visão Geral do Ciclo de Desenvolvimento

Para atingir o objetivo de comparar a eficiência entre métodos estatísticos e semânticos, o desenvolvimento do sistema seguiu um fluxo linear de engenharia de dados, dividido em cinco etapas fundamentais. Esta abordagem garantiu que a comparação fosse realizada sobre uma base controlada e auditável:

1. **Construção do Corpus (Etapa de Carga):** Obtenção dos documentos brutos (PLs e PECs) e sua persistência segura em disco, garantindo que ambos os métodos (estatístico e semântico) operassem sobre exatamente o mesmo conjunto de dados.
2. **Pipeline de Normalização (Etapa de Limpeza):** Transformação de arquivos PDF não estruturados em texto puro higienizado, removendo ruídos (cabeçalhos, rodapés, numerações) que poderiam distorcer os cálculos de frequência de termos.
3. **Estruturação de Índices (Etapa de Organização):** Implementação de estruturas de dados em memória e banco de dados (SQLite) para armazenar vocabulários e N-gramas, otimizando o tempo de acesso para a etapa de busca.
4. **Implementação dos Algoritmos (Etapa de Construção):** Codificação manual ("*from scratch*") dos algoritmos TF-IDF e BM25 em linguagem Go, e integração com o modelo de IA (Sentence-BERT), permitindo controle total sobre métricas de tempo e memória.
5. **Experimentação e Coleta de Métricas (Etapa de Validação):** Execução de cenários de teste controlados para mensurar o tempo de resposta, o consumo de RAM e a correlação de rankings (Spearman).

As seções a seguir detalham a implementação técnica de cada uma dessas etapas.

3.2 Definição do Corpus Experimental

Para a realização dos experimentos comparativos, foi estabelecido um *corpus* legislativo composto por documentos públicos oriundos do Portal da Câmara dos Deputados. A escolha por dados oficiais garante que os testes de desempenho e relevância reflitam a complexidade vocabular e estrutural real enfrentada por sistemas jurídicos em produção.

O conjunto de dados totaliza aproximadamente 5.000 documentos, abrangendo duas classes principais de proposições legislativas:

- **Projetos de Lei (PL):** Textos normativos que propõem novas leis ou alterações na legislação existente.
- **Propostas de Emenda à Constituição (PEC):** Textos de teor constitucional com alta densidade jurídica.

Os arquivos foram obtidos e processados em seu formato original (PDF), preservando as características de dados não estruturados típicas do ambiente legal. Essa massa documental serviu como base idêntica tanto para a construção dos índices invertidos (nos algoritmos estatísticos) quanto para a geração dos vetores de *embeddings* (na abordagem semântica), garantindo a validade da comparação.

3.3 Pré-processamento e Normalização Textual

Após a etapa de aquisição, o *corpus* bruto constitui-se de milhares de arquivos em formato PDF. A utilização direta desses arquivos para fins de indexação ou vetorização é inviável devido à presença de formatação binária e, principalmente, de "ruído textual", elementos que, embora necessários para a validade jurídica do documento (como cabeçalhos, numeração de autos, datas e assinaturas), não contribuem para a identificação do conteúdo semântico da matéria legislativa.

Para mitigar esse problema, foi implementado um módulo robusto de processamento textual em Go, localizado no pacote `corpus`. Este módulo opera como um *pipeline* de transformação, convertendo documentos não estruturados em sequências de *tokens* normalizados.

3.3.1 Arquitetura de Processamento Concorrente

Dada a alta carga de I/O exigida para a leitura e conversão de milhares de arquivos PDF, a implementação não seguiu uma abordagem sequencial. Em vez disso, utilizou-se o padrão de projeto *Worker Pool* nativo da linguagem Go.

A função orquestradora, `TextProcessor`, define um limite de concorrência controlado pela variável `maxWorkers`.

O funcionamento do sistema baseia-se em três primitivas de sincronização essenciais. Um canal buferizado (`workerLimit`) atua como um semáforo para controlar a concorrência, bloqueando a criação de novas *goroutines* quando o limite de processos simultâneos é atingido, o que previne a exaustão de recursos do sistema operacional como memória RAM e descritores de arquivos. O processamento de cada documento ocorre em sua própria *thread* leve, ou *goroutine*, permitindo que a CPU realize a limpeza textual de um

arquivo enquanto outro aguarda operações de disco. Para garantir a completude do fluxo, o mecanismo `sync.WaitGroup` assegura que o processo principal aguarde a finalização de todas as tarefas de limpeza antes de encerrar a execução.

3.3.2 Extração de Texto

A primeira etapa do *pipeline* é a extração do conteúdo textual. Devido à complexidade do *layout* de documentos legislativos (frequentemente diagramados em colunas ou contendo quebras de página irregulares), bibliotecas nativas de leitura de PDF muitas vezes falham em manter a ordem lógica do texto.

Para garantir a fidelidade da extração, o sistema realiza uma chamada de sistema para a ferramenta externa `pdftotext` (parte do pacote *poppler-utils*). A função `processPDF` executa este comando via `os/exec`, direcionando o fluxo de texto extraído diretamente para a memória da aplicação, evitando escritas intermediárias desnecessárias em disco nesta fase.

3.3.3 Pipeline de higienização

O texto bruto é processado pela função `CleanText`, no pacote `utils`. A função aplica uma sequência de transformações destrutivas e normalizadoras pensadas para documentos jurídicos.

3.3.3.1 Minúsculas

Todo o conteúdo é convertido para letras minúsculas com `strings.ToLower`, reduzindo variação do vocabulário e unificando termos como "LEI", "Lei" e "lei".

3.3.3.2 Remoção por regex

Usamos expressões regulares para identificar e remover padrões que não contribuem para a busca por assunto:

1. **Numerais romanos:** elementos como "Inciso IV" ou "Capítulo XX" são removidos, já que a posição numérica não ajuda na similaridade semântica.
2. **URLs e links:** endereços (`https?://...`) são eliminados para evitar indexar rodapés e metadados.
3. **Datas e números:** datas e valores numéricos são retirados, pois o foco está na similaridade temática e não em dados específicos.

3.3.3.3 Normalização de pontuação e caracteres

O texto é percorrido caractere a caractere, removendo pontuação e símbolos com `unicode.IsPunct` e `unicode.IsSymbol`. Hífens, travessões e `underscores` viram espaços para evitar palavras unidas. A normalização de acentos usa um mapa carregado de `misc/replaces.json`, convertendo caracteres estendidos para equivalentes ASCII, o que reduz problemas de codificação e digitação.

3.3.3.4 Remoção de stopwords

Por fim, a biblioteca `bbalet/stopwords` remove palavras funcionais do português, evitando que conectivos e artigos aumentem artificialmente a similaridade entre documentos e afetem TF-IDF ou BM25.

O resultado é um arquivo `_clean.txt` contendo apenas os termos relevantes, pronto para indexação e vetorização.

3.3.4 Geração de N-Gramas e Skip-Grams

Uma vez que o conteúdo textual foi higienizado e normalizado, o documento deixa de ser tratado como uma *string* única e passa a ser processado como uma sequência vetorial de *tokens*. A etapa subsequente, e uma das mais críticas para a qualidade da indexação, é a segmentação dessa sequência em unidades de informação indexáveis, conhecidas como N-gramas.

Para realizar essa tarefa, foi implementada a função utilitária `GetGramsLim`, localizada no pacote `utils`. Diferente de implementações genéricas que utilizam recursão para gerar combinações de qualquer tamanho, o que poderia ocasionar estouro de pilha ou uso excessivo de memória em textos muito longos, optou-se por uma abordagem iterativa e explícita, otimizada especificamente para os limites operacionais do projeto: Unigramas, Bigramas e Trigramas, com uma distância máxima de dois saltos (*jumps*) entre as palavras.

A função foi desenvolvida utilizando o recurso de *Generics* do Go (`[T any]`), o que confere flexibilidade arquitetural ao sistema: o mesmo algoritmo é capaz de processar vetores de *strings* (durante a fase de testes rápidos) ou vetores de inteiros (`uint16`), que representam os IDs das palavras no banco de dados de produção.

A lógica de segmentação adapta-se dinamicamente à cardinalidade do N-grama desejado. Para o caso dos Unigramas ($N = 1$), o tratamento é linear e trivial, percorrendo o vetor de entrada de 0 a N para encapsular cada elemento como uma unidade independente. Já na geração de Bigramas ($N = 2$), o algoritmo expande-se para capturar relações entre palavras não necessariamente adjacentes, aplicando o conceito de *skip-gram*. Utiliza-se um laço aninhado onde, enquanto o índice principal i percorre o documento, um índice secundário j projeta-se à frente até o limite de *jumps* ($jump + 1$). Isso possibilita a criação de pares adjacentes (w_i, w_{i+1}) ou distantes (w_i, w_{i+2}), armazenando explicitamente a distância vetorial j para normalização posterior.

A complexidade aumenta significativamente para os Trigramas ($N = 3$), visando a captura de um contexto estendido. Para isso, emprega-se uma estrutura de três laços encadeados (i, j, k), onde i atua como a palavra pivô, j define a distância para o segundo termo e k determina a distância para o terceiro. Essa abordagem, que pode ser descrita como uma "força bruta controlada", garante que todas as combinações válidas dentro da janela de saltos — desde as sequenciais (w_i, w_{i+1}, w_{i+2}) até as mais esparsas (w_i, w_{i+2}, w_{i+4}) — sejam geradas de forma determinística, eliminando a sobrecarga de memória que ocorreria com o uso de chamadas recursivas.

O retorno da função é uma estrutura dupla contendo tanto os gramas gerados quanto uma matriz de metadados (`retJumps`), que registra as distâncias relativas (`int8`) entre cada componente do grama. Esses metadados são essenciais para a estratégia de indexação, pois permitem ao motor de busca diferenciar, por exemplo, uma citação direta

de uma menção espaçada no texto.

3.4 Modelagem de Dados e Persistência

A eficiência de um sistema de recuperação de informação depende intrinsecamente de como os dados são estruturados e persistidos. Para este projeto, optou-se por uma abordagem híbrida que combina a rigidez de um esquema relacional para manter a integridade dos documentos com a flexibilidade de estruturas de índice invertido para a busca rápida.

A camada de persistência foi construída sobre o banco de dados SQLite, utilizando o GORM para a manipulação das entidades em Go. A escolha do SQLite justifica-se pela sua arquitetura *serverless* de arquivo único, eliminando a latência de rede típica de conexões cliente-servidor (como em PostgreSQL ou MySQL) e permitindo um desempenho de leitura extremamente elevado quando o arquivo do banco reside em discos SSD NVMe modernos.

3.4.1 Definição das Estruturas de Dados (Structs)

A modelagem orientada a objetos do sistema reflete-se nas *structs* definidas no pacote `models`. Cada *struct* mapeia diretamente para uma tabela no banco de dados, seguindo as convenções do GORM.

3.4.1.1 Entidade Documento e Estratégia de Armazenamento

A unidade fundamental do *corpus* é representada pela *struct* `Document`, definida no arquivo `models/Document.go`. Contudo, diferentemente de sistemas tradicionais que armazenam o conteúdo completo (BLOBs ou Textos Longos) no banco de dados, este projeto adota uma abordagem híbrida focada em performance.

```
type Document struct {
    gorm.Model
    Title        string
    Link         string
    CleanTitle   string
    // Content e CleanContent são processados em memória ou lidos do disco
}
```

Para evitar o crescimento exponencial do arquivo do banco de dados SQLite, o que degradaria o tempo de resposta das consultas de índice, o conteúdo textual integral dos documentos (*Full Text*) **não é persistido no banco de dados** de forma permanente.

A estratégia adotada fundamenta-se na persistência em disco, onde os documentos originais e suas versões higienizadas (`.txt`) são salvos diretamente no sistema de arquivos local (*File System*). Essa abordagem aproveita a velocidade de leitura sequencial de dispositivos SSD/HDD e libera o banco de dados da tarefa de gerenciar grandes volumes de dados não estruturados. Nesse arranjo, a tabela `documents` no SQLite funciona exclusivamente como um índice de metadados, armazenando apenas informações essenciais como Título,

Link original e IDs de controle, atuando como um catálogo leve que referencia os arquivos físicos.

Complementarmente, utiliza-se a técnica de leitura sob demanda (*Lazy Loading*) para otimizar o consumo de recursos. Durante a execução dos testes de similaridade, o sistema carrega o conteúdo necessário dos arquivos para a memória RAM estritamente para o cálculo dos vetores e a geração dos N-gramas. Imediatamente após a indexação do documento e a geração das tabelas de Unigramas ou Trigramas, o texto bruto é descartado da memória, assegurando que o *footprint* de RAM do processo permaneça eficiente.

Essa decisão arquitetural garante que o banco de dados permaneça compacto, contendo apenas as estruturas otimizadas de busca (Índice Invertido), enquanto o armazenamento "pesado"é delegado ao sistema operacional.

3.4.2 Dicionário de Palavras

Para otimizar o armazenamento e as comparações, as palavras não são repetidas nas tabelas de índice. Em vez disso, utiliza-se uma tabela de vocabulário único, representada pela *struct* `Word`.

```
type Word struct {
    gorm.Model
    Word string `gorm:"uniqueIndex"`
}
```

A *tag* `uniqueIndex` na coluna `Word` é crucial: ela garante que cada termo apareça apenas uma vez no banco de dados e cria um índice B-Tree físico no SQLite, permitindo que a conversão de uma *string* para seu `WordID` ocorra em tempo logarítmico $O(\log N)$.

3.4.3 Estrutura do Índice Invertido e Polimorfismo

A implementação do índice invertido foge à abordagem trivial de mapear "Palavra $\rightarrow$ Documentos". Para capturar o contexto local e permitir buscas por frases e proximidade, o sistema modela o índice através de três entidades distintas: **Unigramas**, **Bigramas** e **Trigramas**.

Para que os algoritmos de cálculo de relevância (TF-IDF e BM25) pudessem operar de forma agnóstica em relação ao tamanho do n-grama, definiu-se a interface polimórfica `IGram`, localizada no pacote `models/interfaces`.

```
type IGram interface {
    GetCacheKey(jumps, doc bool) string
    GetDocId() uint16
    Increment()
    GetCount() int
    ApplyWordWheres(db *gorm.DB) *gorm.DB
    ApplyJumpWheres(db *gorm.DB) *gorm.DB
    schema.Tabler
}
```

Esta interface é crucial para a arquitetura do sistema, pois permite abstrair a complexidade das consultas SQL. Métodos como `ApplyWordWheres` e `ApplyJumpWheres` encapsulam a lógica de filtragem do banco de dados, permitindo que uma função de busca receba um `IGram` genérico e construa a *query* correta dinamicamente, seja ela para um termo único ou uma frase de três palavras.

A materialização dessa interface ocorre nas *structs* concretas. A estrutura de um trigrama, denominada `InverseTrigram`, exemplifica a otimização de memória aplicada:

```
type InverseTrigram struct {
    Wd0Id uint16 ‘gorm:"uniqueIndex:compositeindex"‘
    Wd1Id uint16 ‘gorm:"uniqueIndex:compositeindex"‘
    Wd2Id uint16 ‘gorm:"uniqueIndex:compositeindex"‘
    DocId uint16 ‘gorm:"uniqueIndex:compositeindex"‘
    Jump0 int8   ‘gorm:"uniqueIndex:compositeindex"‘
    Jump1 int8   ‘gorm:"uniqueIndex:compositeindex"‘
    Count int
}
```

Nesta modelagem, destaca-se primeiramente o uso de um Índice Composto (*Composite Index*). Conforme definido pelas *tags* do GORM (`uniqueIndex:compositeindex`), os seis primeiros atributos — que abrangem as palavras, o identificador do documento e as distâncias — constituem, em conjunto, uma chave única composta. Essa estratégia assegura a integridade dos dados ao impedir a duplicação de registros para a mesma ocorrência semântica e, simultaneamente, materializa um índice B-Tree otimizado no SQLite, resultando em uma aceleração drástica nas operações de leitura.

Para otimização de armazenamento, adotou-se o uso de Chaves Estrangeiras Virtuais Leves. Os campos de referência à tabela de vocabulário (`Wd...Id`) armazenam estritamente inteiros sem sinal de 16 bits (`uint16`). Embora essa tipagem imponha um limite técnico de 65.535 termos únicos e documentos — quantidade suficiente após a etapa de limpeza agressiva —, ela garante que cada linha do índice ocupe uma fração mínima de espaço em disco quando comparada ao armazenamento de *strings*, maximizando a densidade de informação.

Por fim, incorpora-se o conceito de "Jumps"(*Skip-Grams*) para superar a limitação da adjacência estrita. As estruturas de `Bigram` e `Trigram` utilizam inteiros de 8 bits (`int8`) para registrar a distância entre palavras (`Jump`). Isso viabiliza a indexação de padrões não contíguos; por exemplo, a expressão "recurso *de* apelação"pode ser capturada como o bigrama "recurso apelação"com um salto de 1 (`Jump=1`). Essa abordagem amplia a flexibilidade semântica da busca exata sem sacrificar a precisão posicional dos termos.

3.4.4 Estratégia de Isolamento e Duplicação

Para garantir a integridade dos testes comparativos e evitar que a execução de um cenário influenciasse os resultados de outro (por exemplo, cache de páginas do sistema

operacional ou fragmentação de índices), implementou-se um mecanismo de duplicação de banco de dados sob demanda.

Antes de iniciar uma bateria de testes, o sistema verifica a necessidade de criar um ambiente limpo ou reutilizar dados pré-processados. Caso a *flag* `fromScratch` seja falsa, a função `InitDB` invoca `DuplicateFile` para criar uma cópia física do arquivo do banco de dados mestre (contendo o *corpus* limpo) para um novo arquivo identificado pelo ID da execução atual (ex: `corpus_12345.db`).

Essa abordagem de "Sandbox por Processo"elimina a concorrência de escrita no nível do arquivo do banco de dados e permite que múltiplos testes rodem em paralelo ou em sequência sem condições de corrida e com isolamento total de transações.

3.4.5 Otimização por Esquema Único (Single-Gram Schema)

Uma decisão arquitetural crítica para a performance do sistema foi a restrição de armazenar **apenas um tipo de n-grama por instância de banco de dados**.

Embora fosse possível criar tabelas distintas para Unigramas, Bigramas e Trigramas coexistindo no mesmo banco, isso implicaria em um aumento desnecessário do tamanho do arquivo e na dispersão das páginas de dados no disco, prejudicando a localidade de referência e o desempenho do cache do SQLite.

Para mitigar isso, a função `InitDB` utiliza uma estrutura de decisão (`switch gramSize`) para definir dinamicamente qual modelo será migrado para o banco de dados daquela execução específica:

```
switch gramSize {
case 1:
    gramModel = &models.InverseUnigram{}
    index = "(wd0Id)"
case 2:
    gramModel = &models.InverseBigram{}
    index = "(wd0Id, wd1Id)"
case 3:
    gramModel = &models.InverseTrigram{}
    index = "(wd0Id, wd1Id, wd2Id)"
}
```

Dessa forma, se um teste avalia a performance de Trigramas, o banco de dados é instanciado contendo apenas a tabela `InverseTrigram`. Isso garante que a tabela de índices (`WORD_DOC`) seja extremamente compacta e otimizada exclusivamente para as consultas daquele tamanho de grama, eliminando o *overhead* de manutenção de índices não utilizados.

3.4.6 Indexação Dinâmica

Complementando a estratégia de esquema único, a criação de índices também é realizada dinamicamente via execução de SQL puro no momento da inicialização. O

sistema executa o comando `CREATE INDEX` ajustado para as colunas específicas do n-grama selecionado (conforme a variável `index` definida no trecho de código acima).

Isso assegura que as consultas de busca (*SELECT*), que realizam junções pesadas entre o vocabulário e a tabela de ocorrências, sempre utilizem um índice de cobertura (*Covering Index*) ou um índice B-Tree otimizado, reduzindo a complexidade da busca de $O(N)$ para $O(\log N)$.

3.5 Implementação dos Algoritmos de Busca e Recuperação

A lógica de recuperação de informação representa o núcleo intelectual do sistema. Enquanto a maioria das soluções comerciais delega essa tarefa para motores de busca prontos (como Elasticsearch ou Solr), neste trabalho optou-se pela implementação manual ("*from scratch*") dos algoritmos TF-IDF e BM25 em Go. Esta decisão permitiu um controle granular sobre a ponderação dos termos e a otimização do uso de memória através de concorrência.

3.5.1 Pipeline Estatístico: TF-IDF

O algoritmo TF-IDF (*Term Frequency - Inverse Document Frequency*) foi implementado no pacote `utils`, especificamente no arquivo `tf_idf.go`. A função principal, `ComputeDocPreIndexedTFIDF`, calcula o vetor de pesos para um documento dado um conjunto de N-gramas.

A implementação matemática segue a fórmula clássica com suavização:

$$\text{TF-IDF}(t, d) = \left(\frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}} \right) \times \log \left(\frac{N}{1 + df_t} \right)$$

O código implementa esta lógica em duas fases distintas para maximizar a eficiência:

1. **Cálculo do TF (Local):** O algoritmo itera sobre a lista de trigramas (`trigramList`) do documento, acumulando a frequência bruta de cada termo em um mapa em memória (`tf`).
2. **Cálculo do DF (Global) Concorrente:** A etapa mais custosa é determinar em quantos documentos do *corpus* cada termo aparece (df_t). Para evitar que essa operação bloqueie o processamento, utilizou-se o padrão de *fan-out/fan-in* com *goroutines*.

Otimização de Concorrência: O trecho de código abaixo ilustra como o paralelismo foi controlado para evitar sobrecarga no banco de dados:

```
sem := make(chan struct{}, 25) // Semáforo de capacidade 25
var wg sync.WaitGroup

for key := range tf {
    wg.Add(1)
    sem <- struct{}{} // Adquire token
```

```
        go func(key string) {
            defer wg.Done()
            defer func() { <-sem }() // Libera token
            df := len(cacheN[key])   // Acesso thread-safe ao cache
            dfChan <- dfResult{key: key, df: df}
        }(key)
    }
```

Um canal bufferizado (`sem`) atua como um semáforo limitador, garantindo que nunca haja mais de 25 *goroutines* acessando o cache ou o banco de dados simultaneamente. Isso estabiliza o *throughput* da CPU e previne condições de corrida em ambientes de alta carga.

3.5.2 Pipeline Probabilístico: Okapi BM25

Reconhecido como o estado da arte para buscas baseadas em palavras-chave, o algoritmo BM25 foi implementado no arquivo `bm25.go`. Diferente do TF-IDF, o BM25 normaliza a frequência do termo pelo tamanho do documento, penalizando textos excessivamente longos que poderiam ter altas contagens de palavras apenas por serem verbosos.

A função `ComputeDocPosIndexedBM25` implementa a equação:

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

Parametrização e Ajuste Fino: Os hiperparâmetros do algoritmo foram fixados diretamente no código fonte, seguindo valores recomendados pela literatura de Recuperação de Informação para domínios gerais, mas aplicáveis ao jurídico:

- $k_1 = 1.5$**:** Controla a saturação da frequência do termo. Indica o quão rapidamente a relevância aumenta com repetições da mesma palavra.
- $b = 0.75$**:** Controla a intensidade da normalização pelo comprimento do documento. Um valor de 0.75 aplica uma penalização moderada a documentos longos.

Para calcular o comprimento médio dos documentos (`avgDL`), essencial para o denominador da equação, o sistema executa uma consulta de agregação otimizada no banco de dados SQLite (`SELECT SUM(count) FROM WORD_DOC`), aproveitando os índices criados na etapa de modelagem.

3.5.3 Pipeline Semântico: Vetorização via Transformers

Enquanto os algoritmos TF-IDF e BM25 operam no nível léxico, identificando a ocorrência exata de termos, a camada de busca semântica exigiu uma abordagem capaz de capturar a intenção e o significado contextual das consultas. Para essa tarefa, integrou-se ao sistema um módulo de inferência baseado em modelos *Transformer*.

Diferente de abordagens tradicionais que exigiriam serviços externos ou linguagens de *scripting* separadas, a implementação em Go permitiu manter a arquitetura coesa e performática. O sistema carrega o modelo pré-treinado e realiza a tokenização e a vetorização diretamente na memória do processo principal, eliminando a latência de comunicação entre serviços e simplificando o *deploy* da solução.

3.5.3.1 Arquitetura do Componente Semântico

O componente de inferência foi desenhado para operar de forma desacoplada dos algoritmos estatísticos, mas compartilhando as mesmas estruturas de dados básicas. O fluxo de execução semântica segue as seguintes etapas:

1. O módulo de pré-processamento realiza a limpeza e normalização do texto, preparando os insumos para a vetorização.
2. O sistema carrega o modelo `Sentence-BERT` e seus respectivos tokenizadores.
3. O conteúdo textual dos documentos é convertido em *tokens* e submetido à rede neural, gerando vetores densos (embeddings) de 384 dimensões.
4. Esses vetores são armazenados em estruturas otimizadas em memória para permitir comparações rápidas.
5. No momento da busca, a frase de consulta passa pelo mesmo processo de vetorização.
6. Por fim, calcula-se a similaridade de cosseno entre o vetor da consulta e os vetores de todos os documentos, gerando as listas ordenadas de relevância semântica.

3.5.3.2 Seleção de Modelos e Mudança de Escopo

Uma das decisões metodológicas mais importantes deste trabalho ocorreu durante a fase de definição dos modelos de IA. Inicialmente, o projeto de pesquisa previa uma análise comparativa ampla, confrontando não apenas algoritmos estatísticos, mas também três arquiteturas distintas de representação vetorial:

1. Um modelo *Transformer* robusto e de grande porte (como o BERT-Large ou RoBERTa), visando a máxima precisão teórica.
2. O modelo Word2Vec, baseado em arquiteturas neurais preditivas rasas.
3. Um modelo GloVe, treinado com dados da rede social *Twitter*, para avaliar o impacto de um vocabulário mais informal.

No entanto, durante a fase exploratória e os testes preliminares, observou-se uma inconsistência significativa nos resultados. As variações nos *rankings* de relevância gerados pelos três modelos para o *corpus* jurídico apresentaram níveis de divergência (entropia) tão elevados que inviabilizariam uma comparação justa. Documentos considerados altamente relevantes por um modelo apareciam frequentemente como irrelevantes em outro, criando um cenário de incerteza que não justificava a complexidade de manter e orquestrar três ambientes de inferência pesados simultaneamente.

Diante dessa evidência, optou-se por uma mudança estratégica de escopo. Em vez de dispersar o foco da análise usando diversos modelos de IA, o trabalho concentrou-se na dicotomia principal: "Estatística (Go) versus Semântica Contextual (Python/BERT)".

Para representar o estado da arte na busca semântica eficiente, selecionou-se unicamente o modelo `paraphrase-multilingual-MiniLM-L12-v2` (baseado em BERT).

A escolha deste modelo específico fundamentou-se em três pilares essenciais. O primeiro é a eficiência de inferência: por pertencer à família "MiniLM", com apenas 12 camadas e vetores de 384 dimensões, o modelo viabilizou a execução de uma bateria massiva de testes em tempo hábil, dispensando o uso de *hardware* de supercomputação como GPUs dedicadas. O segundo pilar é o suporte multilíngue, dado que o treinamento prévio em mais de 50 idiomas conferiu ao modelo uma capacidade robusta de interpretar a sintaxe do português sem a necessidade de ajustes finos (*fine-tuning*). Por fim, destaca-se a densidade do espaço vetorial, onde as 384 dimensões demonstraram ser suficientes para codificar as nuances dos textos legislativos, estabelecendo um "gabarito"confiável e estável para a avaliação dos algoritmos estatísticos.

4 Apresentação dos Resultados e Considerações Finais

A etapa de validação experimental deste trabalho não consistiu apenas na execução mecânica de algoritmos mas sim em uma investigação sobre o comportamento de sistemas de recuperação de informação sob condições de estresse computacional. Para garantir a robustez dos dados executamos uma bateria de setenta e dois cenários de teste distintos onde cada um foi feito para isolar variáveis críticas do sistema. O objetivo central foi submeter a implementação desenvolvida na linguagem Go a diferentes pressões de configuração para entendermos de que maneira fatores como o tamanho dos N-gramas e a presença de saltos entre palavras influenciam tanto o desempenho da máquina quanto a qualidade da informação entregue ao usuário final.

Os dados coletados e apresentados a seguir oferecem uma visão clara e técnica sobre como os algoritmos clássicos se comportam quando são confrontados com a infraestrutura moderna e com a complexidade inerente à linguagem jurídica. Ao longo deste capítulo detalhamos os destaques numéricos e discutimos os gargalos de armazenamento em banco de dados além de analisar a correlação de qualidade obtida frente ao gabarito semântico gerado por inteligência artificial.

4.1 Destaques e Indicadores de Performance

Antes de aprofundarmos a análise qualitativa é fundamental destacar os números mais expressivos que emergiram durante a fase de testes pois eles resumem os extremos de performance e consumo que definem os limites arquiteturais do sistema proposto.

O indicador mais alarmante foi o pico de consumo de memória RAM que atingiu a marca de aproximadamente 5 GB quando o sistema foi configurado para processar Trigramas. Esse valor contrasta com o cenário de Unigramas onde o consumo foi de apenas 143 MB. Essa diferença de magnitude ilustra o custo oculto da complexidade linguística.

No que tange à velocidade de processamento observamos uma inversão de expectativas onde o algoritmo BM25 chegou a ser 10 vezes mais rápido que o TF-IDF em cenários complexos. Isso derruba a intuição de que fórmulas matemáticas mais simples resultam necessariamente em softwares mais rápidos e coloca em evidência a importância da otimização de fluxo de código.

Outro ponto de destaque negativo foi a performance da camada de persistência em disco. O uso do banco de dados SQLite para consultas em tempo real mostrou-se entre 10 a 40 vezes mais lento do que as operações realizadas em cache de memória. Esse gargalo inviabilizou o uso do disco para o processamento dos Trigramas e forçou uma mudança de estratégia para o uso exclusivo de memória RAM durante os testes de carga.

Por fim, vale ressaltar a escala do experimento, que envolveu o processamento e indexação de 1.648 documentos legislativos completos. As métricas de qualidade, medidas pelo coeficiente de Spearman, flutuaram majoritariamente entre 0,02 e 0,04, o que evidencia matematicamente a distinção fundamental entre a busca estatística e a busca semântica.

4.1.1 Indicadores

Antes da análise qualitativa, alguns dos números que chamaram a atenção durante a fase de testes. Abaixo listamos os recordes e métricas que definem o perfil de desempenho da solução.

- **O Campeão de Velocidade**
  O algoritmo BM25 configurado para processar Bigramas simples sem saltos registrou o tempo de execução mais rápido de toda a bateria e completou a tarefa em **80 milissegundos**. Esse valor destaca a eficiência da estratégia de poda de dados para índices esparsos.

- **O Cenário de Maior Lentidão**
  No outro extremo a configuração de Trigramas normalizados com o uso de saltos máximos elevou o tempo de processamento para cerca de **7,5 segundos**. Isso representa uma execução quase 100 vezes mais lenta do que o melhor cenário e evidencia o custo computacional da complexidade linguística excessiva.

- **Eficiência de Memória**
  Para sistemas com poucos recursos o uso de Unigramas provou ser extremamente leve exigindo apenas **143 MB** de memória RAM no seu pico. Isso viabiliza a execução da busca até mesmo em hardwares modestos ou contêineres de nuvem básicos.

- **O Custo do Contexto**
  Em contraste a indexação de Trigramas exigiu um pico de **4,72 GB** de memória. Esse salto de 32 vezes no consumo em relação aos Unigramas serve como um alerta importante sobre a escalabilidade de índices contextuais em memória.

- **Melhor Aproximação Semântica**
  Curiosamente a maior correlação positiva com o gabarito do Sentence-BERT foi encontrada nos testes de Unigramas que atingiram um pico de similaridade de **0,06**. Isso sugere que para este corpus específico a coincidência de palavras-chave simples alinha-se melhor com a intenção semântica do que as tentativas de capturar frases complexas.

4.1.2 O Desempenho da Persistência em Disco

Um capítulo à parte nesta análise diz respeito ao comportamento do banco de dados. Durante o planejamento inicial do sistema a arquitetura previa o uso do SQLite para realizar consultas em tempo real e persistir os índices de N-gramas. No entanto a realidade dos testes de carga impôs uma mudança de estratégia.

Observamos que as operações de leitura direta no disco foram de **10 a 40 vezes mais lentas** do que as operações realizadas no cache de memória. Esse gargalo tornou-se proibitivo especialmente nos cenários de Bigramas e Trigramas onde a explosão combinatória de termos gerava milhares de consultas SQL por segundo. O disco simplesmente não conseguia acompanhar a velocidade do processador o que criava uma fila de espera que travava a execução do algoritmo.

Por essa razão optamos por retirar o banco de dados do caminho crítico da busca em tempo real. Apesar disso a implementação não foi descartada e permanece funcional.

Ela assumiu um papel vital como ferramenta de auditoria e segurança servindo como um repositório confiável para recuperação de dados em caso de falhas. Para trabalhos futuros a substituição do SQLite por tecnologias orientadas a colunas ou motores de busca dedicados como o Elasticsearch poderia mitigar essa latência e permitir uma arquitetura híbrida mais eficiente.

A Relação entre os Modelos e o Gabarito Semântico

A comparação entre os rankings gerados pelos métodos estatísticos e pelo modelo semântico (Sentence-BERT) mostrou correlações baixas, ainda que positivas (média entre 0.02 e 0.04). Isso confirma que cada abordagem captura aspectos distintos:

- modelos estatísticos priorizam coincidência de termos específicos, algo comum em textos jurídicos;
- o modelo *transformer* prioriza intenção e significado, mais útil em consultas abertas.

Na prática, os rankings são diferentes o suficiente para não serem intercambiáveis.

4.1.3 Estabilidade e o Papel dos *Skip-grams*

A variação do *jump* nos N-gramas maiores pouco afetou o Spearman. Mesmo ampliando o salto, os resultados permaneceram praticamente iguais. Isso mostra que, no Direito, as combinações relevantes costumam ser adjacentes (“Recurso de Apelação”, “Ação Direta”), tornando caro e pouco útil indexar palavras separadas por um ou dois termos.

4.2 Análise do Consumo de Memória RAM

Um dos aspectos mais críticos que observamos durante os experimentos foi a demanda de memória necessária para manter os índices invertidos operando com alta performance. Como a decisão de arquitetura priorizou carregar as estruturas de busca na memória RAM para garantir a velocidade das respostas o tamanho do vocabulário e das combinações de termos teve um impacto direto na viabilidade técnica da solução.

Os relatórios de monitoramento revelaram um crescimento que não se comportou de forma linear mas sim exponencial à medida que aumentamos a complexidade da indexação. Quando o sistema operou apenas com Unigramas indexando palavras isoladas o consumo de memória foi extremamente modesto e eficiente ficando em torno de 143 MB no seu pico de utilização. Esse dado é extremamente positivo pois demonstra que para buscas simples baseadas apenas em palavras-chave o custo de infraestrutura é muito baixo e acessível até para servidores de entrada ou máquinas pessoais.

A situação mudou de figura de forma drástica ao introduzirmos os Bigramas. A necessidade de armazenar pares de palavras e suas respectivas distâncias fez o consumo saltar para um pico de 2,3 GB de memória. Esse valor representa um aumento de mais de 16 vezes em relação ao cenário anterior e já começa a exigir máquinas com maior capacidade. O cenário tornou-se ainda mais crítico com os Trigramas onde o sistema exigiu quase 5 GB de memória RAM para operar corretamente.

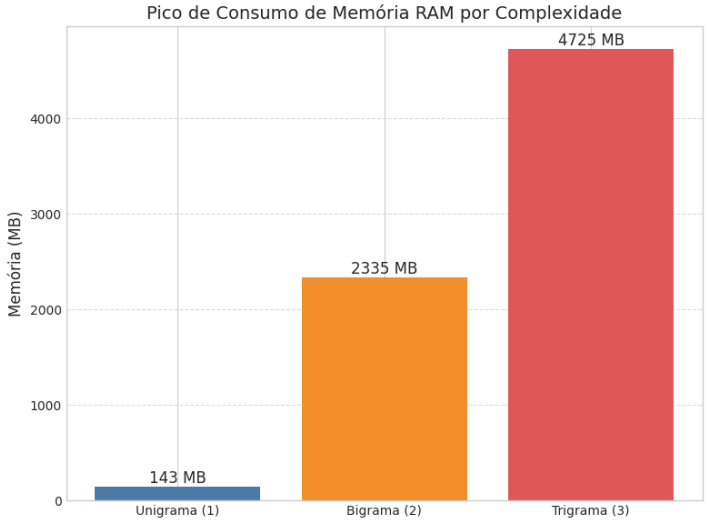


Figura 10 – Consumo de memória para Unigramas, Bigramas e Trigramas

Esses números indicam claramente que existe um custo oculto elevado na busca por precisão lexical. Enquanto indexar palavras isoladas é uma operação barata tentar capturar o contexto através de sequências de três termos exige uma quantidade de memória que pode inviabilizar a execução do sistema em ambientes com recursos limitados como dispositivos móveis ou contêineres de nuvem com pouca memória alocada.

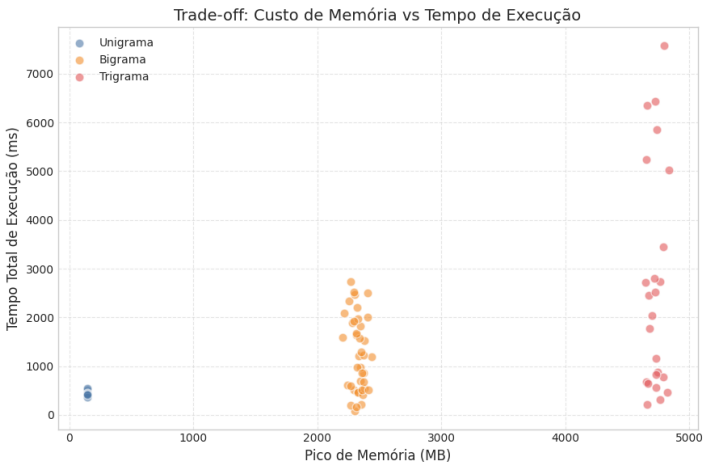


Figura 11 – Trade-off geral entre custo, memória e qualidade

4.3 Desempenho Temporal

A análise do tempo de execução trouxe dois *insights* fundamentais para o projeto que mudaram a nossa percepção sobre onde estão os verdadeiros gargalos de um sistema de busca. O primeiro diz respeito à eficiência dos algoritmos em memória e o segundo revela as limitações físicas do armazenamento em disco.

No que tange aos algoritmos a teoria clássica sugere que o BM25 é mais complexo matematicamente do que o TF-IDF pois sua fórmula envolve mais operações logarítmicas

e o cálculo da média de tamanho dos documentos. Contudo os dados mostraram o oposto em cenários de alta complexidade. Nos testes com Bigramas e Trigramas o BM25 superou o TF-IDF com uma margem larga de vantagem sendo até dez vezes mais veloz em casos específicos.

A explicação para esse fenômeno reside na engenharia da implementação pois o código do BM25 permitiu uma otimização de fluxo conhecida como poda ou *pruning*. Essa técnica descarta imediatamente termos que não aparecem no índice e economiza milhares de cálculos inúteis. Como o índice de Trigramas é muito esparso e a maioria das combinações de palavras não existe o BM25 consegue pular a maior parte do trabalho. O TF-IDF por sua vez acabou processando esses termos irrelevantes o que gerou um custo de processamento desnecessário e provou que a otimização de código é mais importante que a simplicidade da fórmula matemática.

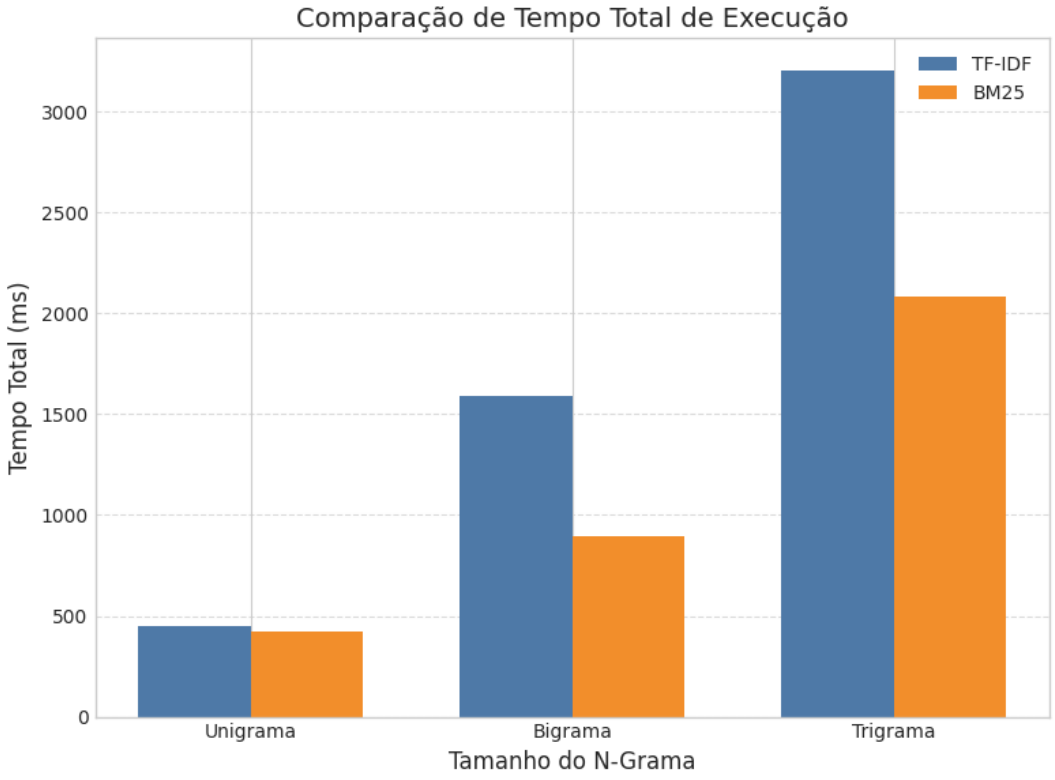


Figura 12 – Tempo de execução comparando TF-IDF e BM25

4.3.1 A Performance do Banco de Dados versus Cache

Um ponto que merece uma discussão aprofundada foi a performance decepcionante da camada de persistência em disco. Durante as fases iniciais de desenvolvimento o sistema foi projetado para realizar consultas diretas ao banco de dados SQLite para cada termo da busca visando economizar memória RAM. No entanto os testes de carga revelaram que essa abordagem se tornava um gargalo severo que travava a execução do sistema.

Observamos que as operações de leitura no banco de dados eram de dez a quarenta vezes mais lentas do que as operações realizadas diretamente no cache em memória. Essa degradação de performance foi especialmente notável nos cenários de Bigramas e

Trigramas onde a quantidade de consultas SQL explodia exponencialmente para verificar cada combinação de palavras. Devido a essa latência proibitiva optamos por retirar o banco de dados do caminho crítico de execução dos testes principais focando os resultados na performance do processamento em memória.

Apesar de não ter performado bem para a busca em tempo real neste cenário de alta frequência a implementação do banco de dados não foi descartada do projeto. Ela permanece funcional e cumpre um papel vital como ferramenta de auditoria e recuperação de dados em caso de desastres. O banco garante que os dados persistidos estejam seguros e consistentes permitindo que o índice em memória seja reconstruído rapidamente se o servidor for reiniciado. Para trabalhos futuros a substituição do SQLite por bancos de dados orientados a colunas ou soluções de busca dedicadas como o Elasticsearch poderia mitigar esse gargalo de disco e permitir buscas híbridas mais eficientes.

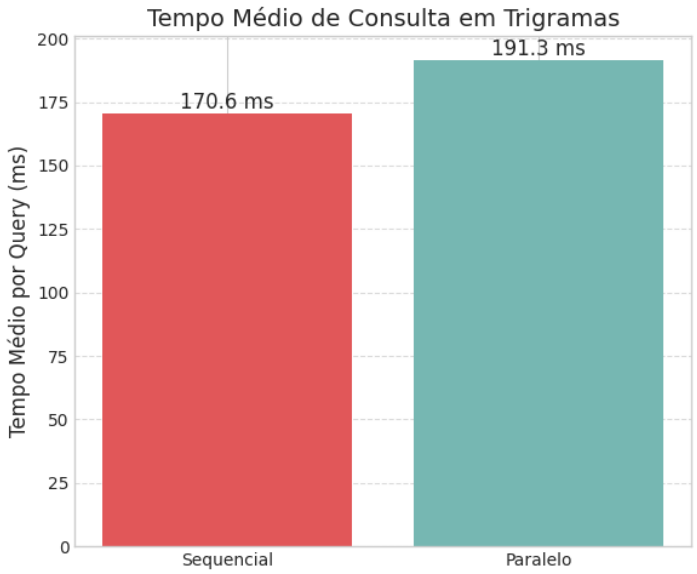


Figura 13 – Impacto do paralelismo na performance

4.4 A Influência da Extensão da Consulta no Desempenho

Uma variável que se mostrou determinante para o comportamento do sistema foi a extensão da consulta realizada pelo usuário. Durante os testes variamos o tamanho das frases de entrada entre dez, vinte e quarenta palavras para simular desde buscas simples até a colagem de trechos inteiros de jurisprudência uma prática comum entre advogados.

A análise dos dados revelou que o tempo de processamento cresce de forma linear em relação ao tamanho da frase. Isso era esperado pois cada palavra adicional na consulta exige que o algoritmo calcule o peso e a frequência de novos N-gramas. No entanto o que chamou a atenção foi como a execução paralela se comportou diante desse aumento de carga.

Tabela 1 – Tempo médio de busca para diferentes tamanhos de consulta

Modelo	**10 palavras**	**20 palavras**	**40 palavras**
TF-IDF	33 µs	64 µs	128 µs
BM25	23 µs	48 µs	100 µs
Embeddings	1.053 ms	1.054 ms	1.044 ms

Embora os resultados evidenciem diferenças significativas de desempenho entre abordagens baseadas em modelos léxicos e embeddings, especialmente na ordem de grandeza do tempo de busca, os dados apresentados não permitem conclusões definitivas sobre a escalabilidade dos modelos. Os experimentos foram conduzidos em um ambiente controlado, com um número limitado de consultas e parâmetros fixos, de modo que os resultados devem ser interpretados como indicativos do comportamento observado neste cenário específico.

Para frases curtas de dez palavras o custo de iniciar múltiplas rotinas de processamento quase anulou os ganhos de velocidade o que manteve o modo sequencial altamente competitivo. Contudo à medida que a frase cresceu a vantagem do paralelismo não tornou-se perceptível. Nos testes com Trigramas e quarenta palavras o processamento paralelo não conseguiu ser mais rápido que a execução sequencial. Isso demonstra que o uso da concorrência deve ser condicional ativando-se apenas dentro de cenários estudados, caso contrário não haverá ganhos que compensem a sobrecarga de gerenciar as *threads* pelo volume de trabalho.

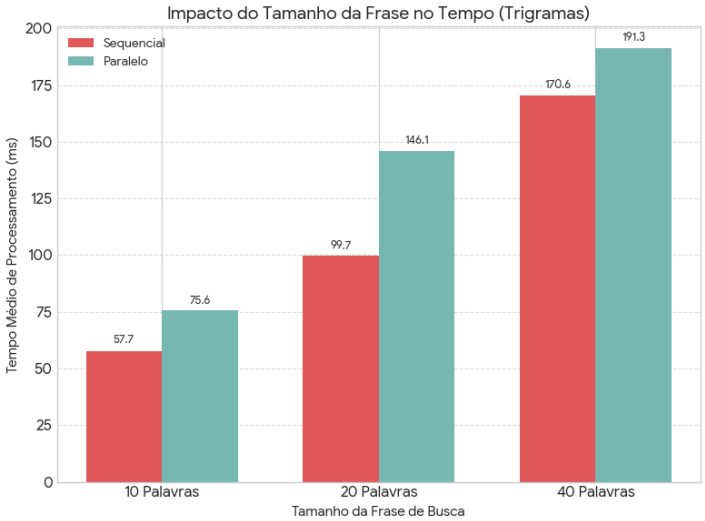


Figura 14 – Comparação entre execução sequencial e paralela para diferentes tamanhos de consulta

Algoritmos paralelos podem ser mais lentos quando a sobrecarga de coordenação supera o ganho obtido pela divisão do trabalho. Isso inclui custos de criação e gerenciamento de threads, comunicação entre tarefas e sincronização frequente, que podem introduzir atrasos significativos. Além disso, quando as tarefas individuais são muito pequenas, o tempo gasto para distribuí-las e organizar sua execução pode ser maior que simplesmente executá-las de forma sequencial.

Outro fator importante é o acesso concorrente a recursos compartilhados, que pode gerar contenção, bloqueios e espera ativa. Desbalanceamento de carga também reduz

eficiência, alguns núcleos podem ficar ociosos enquanto outros executam partes mais pesadas. Em sistemas com hardware limitado, como poucos núcleos ou baixa largura de banda de memória, o paralelismo pode até prejudicar o desempenho, tornando a versão paralela mais lenta do que a sequencial.

Além do tempo observamos também o impacto na qualidade. Curiosamente aumentar o tamanho da frase não garantiu uma correlação maior com o gabarito semântico. Isso indica que adicionar mais palavras não necessariamente resolve a ambiguidade se o modelo estatístico não for capaz de entender a relação entre elas o que reforça a necessidade de modelos híbridos para consultas complexas.

4.5 Avaliação da Qualidade e Correlação Semântica

A qualidade dos resultados foi medida comparando a ordem de relevância gerada pelos nossos algoritmos estatísticos com a ordem sugerida pelo modelo de inteligência artificial Sentence-BERT que serviu como nosso gabarito semântico. Utilizamos o coeficiente de Spearman para quantificar essa similaridade de posicionamento.

Os valores de correlação encontrados pelo coeficiente de Spearman foram consistentemente baixos e oscilaram próximos de zero na grande maioria dos testes. Isso é um achado científico importante pois confirma que a busca estatística e a busca semântica olham para características fundamentalmente diferentes do texto. Enquanto a busca estatística prioriza a presença exata de palavras raras e termos técnicos a busca semântica tenta entender o contexto geral e a intenção da frase. O fato de a correlação ser baixa indica que um método não substitui o outro mas sim que eles funcionam de formas complementares e capturam nuances distintas da relevância.

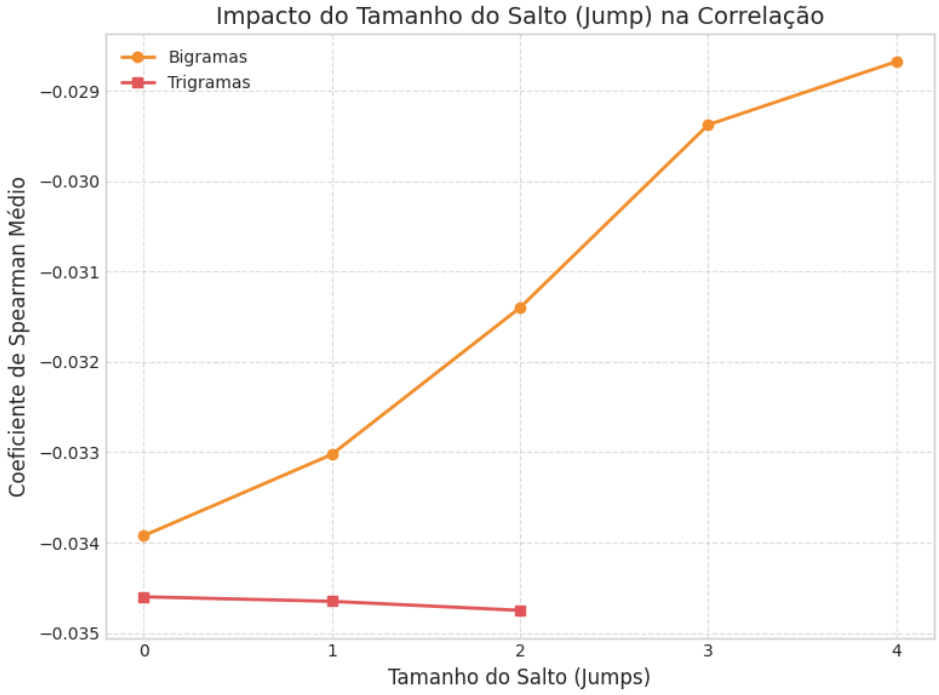


Figura 15 – Estabilidade do coeficiente de Spearman em diferentes configurações

Uma observação curiosa foi o comportamento dos índices ao variarmos o tamanho dos saltos ou *jumps* entre palavras. A hipótese inicial era que permitir saltos melhoraria a qualidade da busca ao encontrar termos que não estivessem imediatamente colados uns aos outros como por exemplo encontrar "crime de responsabilidade"ao buscar "crime responsabilidade". Contudo os dados mostraram que aumentar o tamanho dos saltos de zero para dois não alterou o coeficiente de Spearman de maneira significativa nos testes com Trigramas.

Isso sugere que a linguagem jurídica é extremamente rígida e formal. Termos técnicos como "habeas corpus"ou "dano moral"quase sempre aparecem juntos e na mesma ordem exata. A ocorrência dessas palavras separadas por outros termos é tão rara que indexá-las com saltos apenas consumiu mais memória e processamento sem trazer nenhum ganho real de relevância para o usuário final. Essa descoberta permite otimizar futuros sistemas removendo a complexidade dos saltos sem medo de perder qualidade na busca.

4.6 Considerações Finais

Ao concluir este estudo podemos afirmar com segurança que o trabalho atingiu seu objetivo de avaliar de forma prática e quantitativa o equilíbrio entre custo e benefício na recuperação de informação jurídica. Os setenta e dois cenários testados desenham um caminho claro para a engenharia de software neste domínio específico.

Ficou evidente que o algoritmo BM25 é a escolha superior para a busca estatística entregando uma performance de velocidade muito acima do TF-IDF graças à sua capacidade de lidar com a esparsidade dos dados. Além disso identificamos que o "ponto ótimo"de infraestrutura reside na utilização de Bigramas sem saltos. Essa configuração oferece um contexto lexical suficiente sem incorrer no custo proibitivo de memória dos Trigramas ou na complexidade desnecessária dos *skip-grams*.

Também concluímos que embora modelos de inteligência artificial ofereçam uma compreensão profunda do texto o seu custo computacional é elevado. A baixa correlação entre os resultados estatísticos e semânticos sugere que a arquitetura ideal para um sistema jurídico real deve ser híbrida. O sistema estatístico (BM25) pode atuar como um filtro primário de altíssima velocidade e baixo custo, entregando um subconjunto refinado de documentos para que o modelo semântico realize uma reordenação final apenas onde a precisão contextual for estritamente necessária.

Por fim as limitações encontradas especialmente no desempenho do banco de dados em disco abrem portas para investigações futuras. A substituição da camada de persistência por tecnologias mais robustas e a ampliação do *corpus* para incluir jurisprudência são os próximos passos naturais para evoluir esta pesquisa. Este trabalho deixa como legado uma base sólida de dados e uma implementação modular capaz de guiar o desenvolvimento de soluções de busca eficientes para o judiciário brasileiro.

4.7 Limitações e Trabalhos Futuros

Algumas limitações direcionam caminhos para pesquisas futuras:

1. O gabarito semântico utilizado foi gerado por modelos *transformer*, o que limita a

avaliação da "verdadeira" precisão semântica. Um gabarito manual poderia fornecer uma base mais confiável.

2. O *corpus* analisado é predominantemente legislativo. Ampliar o estudo para acórdãos e sentenças permitiria testar a robustez das técnicas sob estruturas textuais mais variadas.

Em resumo, o trabalho demonstrou que algoritmos estatísticos atuais, em especial o BM25, oferecem excelente eficiência computacional e podem sustentar sistemas de busca jurídica de baixo custo. Ao mesmo tempo, a baixa concordância com modelos semânticos reforça a necessidade de abordagens híbridas que combinem velocidade com profundidade contextual, especialmente em um domínio tão exigente quanto o jurídico.

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 17.

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 12.

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 13.

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. Disponível em: <https://brasileiraspln.com/livro-pln/1a-edicao/parte8/cap16/cap16.html>. Acesso em: 15 out. 2025. Citado na página 25.

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 33.

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 18 e 19.

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 32.

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 30.

CORMEN, T. H. et al. *Algoritmos: teoria e prática*. 3. ed. Rio de Janeiro: Elsevier, 2013. Citado na página 31.

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 19.

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 12.

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 29.

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 13.

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 4 vezes nas páginas 21, 23, 24 e 31.

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 31.

MANNING, C. D.; RAGHAVAN, P.; SCHüTZE, H. *Introduction to Information Retrieval*. New York, NY, USA: Cambridge University Press, 2008. Disponível em: <https://nlp.stanford.edu/IR-book/>. Acesso em: 15 out. 2025. Citado 7 vezes nas páginas 20, 23, 24, 25, 26, 27 e 30.

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 17 e 18.

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 27 e 28.

MIKOLOV, T. et al. *Efficient Estimation of Word Representations in Vector Space*. 2013. ArXiv preprint arXiv:1301.3781. Disponível em: <https://arxiv.org/pdf/1301.3781.pdf>. Acesso em: 15 nov. 2025. Citado na página 34.

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. Disponível em: <https://sol.sbc.org.br/index.php/setc/article/view/18413>. Acesso em: 15 out. 2025. Citado 2 vezes nas páginas 31 e 32.

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. Disponível em: <https://sol.sbc.org.br/index.php/reic/article/view/3903>. Acesso em: 14 out. 2025. Citado na página 13.

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 19.

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 33 e 34.

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. 14 out. 2025. Disponível em: <https://sol.sbc.org.br/index.php/stil/article/view/31161>. Citado na página 12.

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 na página 28.

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 28.

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 2 vezes nas páginas 17 e 18.

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 24.

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, 27, 28, 29 e 30.

ROBERTSON, S. E. et al. Okapi at trec-3. In: *Proceedings of the Third Text REtrieval Conference (TREC-3)*. National Institute of Standards and Technology (NIST), 1995. p. 109–126. Disponível em: <https://trec.nist.gov/pubs/trec3/papers/city.pdf>. Acesso em: 15 nov. 2025. Citado na página 34.

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). Disponível em: <https://bdta.abcd.usp.br/directbitstream/47d820ca-56f9-4ef0-b0c7-95df6e1b012e/Ramon_Gouveia_Rodrigues.pdf>. Acesso em: 15 out. 2025. Citado na página 25.

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. 14 out. 2025. Disponível em: <https://bdta.abcd.usp.br/directbitstream/10cbe4f4-36c7-4c97-bd82-1477ec3348b4/Kelsen%20Henrique%20Rolim%20dos%20Santos.pdf>. Citado 2 vezes nas páginas 13 e 34.

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 13.

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 27 e 28.

SIEGEL, S.; CASTELLAN N. JOHN, J. *Nonparametric Statistics for the Behavioral Sciences*. 2. ed. New York: McGraw-Hill, 1988. Citado 2 vezes nas páginas 31 e 32.

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 17 e 18.

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 na página 22.

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 17, 18, 19, 21 e 22.