UNIVERSIDADE DE SÃO PAULO
Instituto de Ciências Matemáticas e de Computação

Rastreando a Evolução Tecnológica: Um Estudo Baseado na Produção Acadêmica da USP

Pedro Augusto Vara

Trabalho de Conclusão de Curso
MBA em Inteligência Artificial e Big Data

ICMC USP
SÃO CARLOS

UNIVERSIDADE DE SÃO PAULO
Instituto de Ciências Matemáticas e de Computação

Rastreando a Evolução Tecnológica: Um Estudo Baseado na Produção Acadêmica da USP

Pedro Augusto Vara

USP - São Carlos

2025

Pedro Augusto Vara

Rastreando a Evolução Tecnológica: Um Estudo Baseado na Produção Acadêmica da USP

Trabalho de conclusão de curso apresentado ao Departamento de Ciências de Computação do Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo - ICMC/USP, como parte dos requisitos para obtenção do título de Especialista em Inteligência Artificial e Big Data.

Área de concentração: Inteligência Artificial.

Orientador: Prof. Dr. Alneu de Andrade Lopes

USP - São Carlos

2025

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados inseridos pelo(a) autor(a)

Vara, Pedro Augusto
V287r Rastreando a Evolução Tecnológica: Um Estudo Baseado na Produção Acadêmica da USP / Pedro Augusto Vara; orientador Alneu de Andrade Lopes. -- São Carlos, 2025.
90 p.

Trabalho de conclusão de curso (MBA em Inteligência Artificial e Big Data) -- Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2025.

1. Inteligência Artificial. 2. Análise de Tendências. 3. Utilização e Abandono de Algorítmos . I. de Andrade Lopes, Alneu, orient. II. Título.

Bibliotecários responsáveis pela estrutura de catalogação da publicação de acordo com a AACR2:
Gláucia Maria Saia Cristianini - CRB - 8/4938
Juliana de Souza Moraes - CRB - 8/6176

DEDICATÓRIA

Dedico este trabalho ao meu esposo, Leandro Fernandes Vara, cuja presença e apoio incondicional foram essenciais em toda esta jornada.

A você, que esteve ao meu lado com paciência, compreensão e incentivo constantes, oferecendo força nos momentos de desafio e serenidade nos períodos de maior exigência.

Sua confiança e encorajamento foram determinantes para que este projeto se concretizasse.

Mais do que um companheiro, você é um verdadeiro exemplo de parceria, amor e generosidade.

Expresso aqui minha profunda gratidão por compartilhar comigo cada passo deste caminho, com dedicação, respeito e carinho.

AGRADECIMENTOS

Aos meus pais, com todo o meu amor e gratidão.
A vocês, que sempre me ofereceram as condições e oportunidades para estudar, aprender e crescer, plantando desde cedo em mim o valor do conhecimento como caminho para a transformação.
Em especial à minha mãe, que foi e continua sendo minha maior incentivadora. Desde a infância, ela me ensina que estudar é o melhor investimento que se pode fazer na vida.
Durante todo o MBA, acompanhou cada etapa, perguntando mês a mês como estavam as aulas, os trabalhos e o andamento do curso — e, antes mesmo deste terminar, já queria saber qual seria o próximo desafio acadêmico.
Esse trabalho é, em grande parte, fruto do amor, da dedicação e do exemplo que recebi de vocês. Levo comigo o que me ensinaram: que aprender nunca é demais e que o conhecimento é um legado que ninguém pode tirar.

A Tomaz Machado, meu fiel escudeiro durante esta jornada.
Foi ele quem, com paciência e generosidade, dedicou até tempo aos seus domingos e parte dos intervalos de almoço para ouvir minhas ideias, esclarecer dúvidas e contribuir com soluções sempre precisas.
Sua parceria e disposição em compartilhar conhecimento fizeram dele praticamente meu coautor nesta caminhada acadêmica.

Agradeço também a todos os amigos que, com paciência e carinho, suportaram minhas conversas intermináveis sobre o TCC, ouviram dúvidas e me motivaram a não desistir.
Em especial, agradeço a Barbara Paiva, que me trouxe calma e lucidez nos dias mais críticos, e à Lívia, Tainá, Fernanda e Pamela, que sempre me lembraram da importância de acreditar no processo e celebrar cada pequena conquista.

EPÍGRAFE

A ciência progride não apenas ao criar novos conhecimentos, mas ao reconhecer os padrões que ligam o que já sabemos.

Gregory Bateson (1979)

RESUMO

VARA, P. A. Tendências tecnológicas em trabalhos de conclusão de curso da USP com uso de Inteligência Artificial e Big Data. 2025. 103 f. Trabalho de Conclusão de Curso (MBA em Inteligência Artificial e Big Data) – Centro de Ciências Matemáticas Aplicadas à Indústria, Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, 2025.

Este trabalho tem como objetivo identificar e analisar as tendências tecnológicas presentes em trabalhos de conclusão de curso (TCCs) disponibilizados na base de dados da Universidade de São Paulo (USP), de modo a compreender a evolução dos temas científicos e tecnológicos ao longo do tempo. Para isso, foi desenvolvido um pipeline de análise textual automatizado, implementado em Python, capaz de extrair, limpar e processar o conteúdo de milhares de arquivos em formato PDF. O método envolve a aplicação de técnicas de Processamento de Linguagem Natural (PLN), stemming, filtragem por wordlist controlada, geração de n-grams e cálculo de similaridade por coeficiente de Dice, com o apoio das bibliotecas PyMuPDF, NLTK e WordCloud. Os resultados obtidos foram agrupados em intervalos de cinco anos, permitindo visualizar o surgimento, a consolidação e o declínio de termos e tecnologias ao longo de diferentes períodos. As visualizações e wordclouds geradas demonstraram a crescente presença de temas relacionados à inteligência artificial, aprendizado de máquina e ciência de dados, evidenciando a consolidação dessas áreas na produção acadêmica recente. Conclui-se que a metodologia proposta é eficiente para identificar padrões evolutivos em acervos acadêmicos e pode ser expandida para outros repositórios de pesquisa, contribuindo para estudos sobre a dinâmica da inovação e da difusão tecnológica.

Palavras-chave: Inteligência Artificial. Big Data. Processamento de Linguagem Natural. Tendências Tecnológicas. Análise de Texto.

ABSTRACT

VARA, P. A. Technological Trends in Undergraduate Theses from the University of São Paulo Using Artificial Intelligence and Big Data. 2025. 103 f. Final Project (MBA in Artificial Intelligence and Big Data) – Center for Applied Mathematical Sciences to Industry, Institute of Mathematical and Computer Sciences, University of São Paulo, São Carlos, 2025.

This study aims to identify and analyze technological trends found in undergraduate theses from the University of São Paulo (USP), seeking to understand the evolution of scientific and technological topics over time. To achieve this, an automated text analysis pipeline was developed in Python, capable of extracting, cleaning, and processing the content of thousands of PDF documents. The method applies techniques of Natural Language Processing (NLP), stemming, controlled wordlist filtering, n-gram generation, and Dice similarity coefficient calculation, supported by libraries such as PyMuPDF, NLTK, and WordCloud. The results were grouped into five-year intervals, allowing visualization of the emergence, consolidation, and decline of technologies and concepts across different periods. The generated wordclouds and graphical analyses revealed the growing presence of topics related to artificial intelligence, machine learning, and data science, highlighting the consolidation of these areas in recent academic production. The proposed methodology proved effective for identifying evolutionary patterns in academic collections and can be extended to other research repositories, contributing to studies on innovation dynamics and technological diffusion.

Keywords: Artificial Intelligence. Big Data. Natural Language Processing. Technological Trends. Text Analysis.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AI – Artificial Intelligence (Inteligência Artificial)

BDTA – Biblioteca Digital de Teses e Dissertações da Universidade de São Paulo

BERT – Bidirectional Encoder Representations from Transformers, modelo de linguagem pré-treinado

Big Data – Conjunto de técnicas para análise de grandes volumes de dados

Dice coefficient – Métrica de similaridade entre cadeias de caracteres utilizada para correspondência aproximada

JSON – JavaScript Object Notation, formato de intercâmbio de dados utilizado na exportação dos resultados

LLM – Large Language Model, modelo de linguagem de larga escala

Machine Learning – Aprendizado de Máquina

N-gram – Sequência de n palavras consecutivas em um corpus textual

NLTK – Natural Language Toolkit, biblioteca Python para Processamento de Linguagem Natural

PLN – Processamento de Linguagem Natural

Python – Linguagem de programação utilizada para desenvolvimento do pipeline de análise

PyMuPDF – Biblioteca Python utilizada para extração textual de arquivos PDF

Stemming – Processo de redução de palavras às suas raízes morfológicas

TF-IDF – Term Frequency–Inverse Document Frequency, métrica de relevância de termos em um corpus

SUMÁRIO

1 INTRODUÇÃO

1.1 Motivação e Contextualização

O avanço tecnológico constitui um elemento fundamental no desenvolvimento da sociedade moderna, promovendo mudanças profundas na forma como indivíduos, empresas e governos operam diariamente. Compreender essa evolução é uma necessidade estratégica para organizações que buscam inovação e vantagem competitiva, além de representar uma exigência acadêmica e científica na análise das transformações socioeconômicas e tecnológicas contemporâneas.

Como destaca Domingos (2015), compreender a evolução histórica dos algoritmos é fundamental para prever os rumos da IA e distinguir as ideias que realmente se consolidaram das que ficaram pelo caminho.

Nesse cenário, ganham destaque tecnologias emergentes como a Inteligência Artificial (IA) e o *Big Data*, que vêm revolucionando diversos setores por meio da análise em larga escala de informações complexas e heterogêneas. A aplicação dessas tecnologias viabiliza desde a automação inteligente de processos até a geração de *insights* estratégicos, fundamentais para uma tomada de decisão mais eficaz e embasada.

De acordo com o Instituto de Tecnologia e Sociedade do Rio de Janeiro (2023), tecnologias como a Inteligência Artificial e o *Big Data* estão remodelando profundamente os processos sociais, econômicos e comunicacionais, o que exige novas abordagens de análise e compreensão para acompanhar essa transformação.

Segundo Han, Pei e Kamber (2011, p. 8), “o processo de transformar dados brutos em *insights* acionáveis requer múltiplas etapas: coleta, limpeza, transformação, modelagem e, por fim, interpretação para tomada de decisão”. Com base nessa perspectiva, o ciclo inicia-se com a preparação dos dados brutos, que envolve coleta, limpeza e organização das informações para torná-las adequadas à análise. Em seguida, realiza-se o treinamento dos modelos de aprendizado de máquina com esses dados processados, permitindo que aprendam padrões e gerem conhecimento útil. Por fim, os modelos treinados são aplicados para extrair evidências ou *insights* que apoiarão na tomada de decisões e que podem se retroalimentar continuamente com novos dados e aprimoramentos do modelo.

Segundo Cambria e White (2014), técnicas de Processamento de Linguagem Natural possibilitam a identificação de tendências, sentimentos e tópicos emergentes em grandes volumes de texto, o que torna possível gerar *insights* relevantes para tomada de decisão.

Segundo Barbosa e Guimarães (2020), o Processamento de Linguagem Natural (PLN) tem sido amplamente adotado nos estudos métricos da informação, justamente por sua capacidade de identificar padrões, estruturas e tendências emergentes em grandes volumes de literatura científica.

Particularmente no contexto acadêmico, a análise da evolução tecnológica por meio de técnicas como o Processamento de Linguagem Natural (NLP) – com destaque para algoritmos de *stemming* e *N-grams* – mostra-se extremamente relevante. Essas técnicas permitem extrair e interpretar informações valiosas de grandes volumes de texto, viabilizando a identificação de padrões e tendências tecnológicas ao longo do tempo.

Um aspecto motivador para a realização deste trabalho foi a ampla disponibilidade pública das informações acadêmicas oferecidas pela Universidade de São Paulo (USP).
A instituição mantém a Biblioteca Digital de Trabalhos Acadêmicos (BDTA), um repositório aberto e acessível que reúne mais de 17 mil arquivos em formato PDF de monografias, teses e dissertações. Essa transparência e acesso facilitado aos dados representa uma oportunidade única para pesquisas baseadas em grandes volumes de texto acadêmico, especialmente aquelas que buscam identificar padrões e tendências ao longo do tempo.

Centralizando o objeto de estudo nos trabalhos produzidos pela USP e analisando como elas evoluíram ao longo para identificar quais áreas tecnológicas ganharam força, quais se consolidaram e quais foram abandonadas ou se tornaram obsoletas.

Ao examinar essa trajetória, é possível obter *insights* valiosos sobre as tendências emergentes e os direcionamentos futuros da pesquisa e da inovação. Além disso, esse estudo reflete a produção científica da USP e contribui para a identificação de lacunas e oportunidades, tanto para a comunidade acadêmica quanto para a formulação de políticas públicas voltadas ao desenvolvimento tecnológico.

1.2 Problema de Pesquisa e Objetivos

Apesar da vasta produção acadêmica disponibilizada pela USP por meio da BDTA, ainda são escassos os estudos que utilizam técnicas de Processamento de Linguagem Natural (PLN) para analisar, de forma sistemática e automatizada, a evolução das tecnologias abordadas ao longo dos anos nesses trabalhos.

O grande volume e diversidade de temas presentes nesses documentos tornam inviável uma análise manual eficiente, o que levanta a seguinte questão central: quais tecnologias têm

sido mais recorrentes, emergentes ou decadentes na produção acadêmica da USP ao longo do tempo, e como essas tendências podem ser identificadas por meio de algoritmos.

Este trabalho tem como objetivo desenvolver um modelo de análise de tendências tecnológicas a partir do conteúdo textual de monografias disponibilizadas na Biblioteca Digital de Trabalhos Acadêmicos (BDTA) da Universidade de São Paulo (USP). A proposta consiste em aplicar técnicas de Processamento de Linguagem Natural (PLN) sobre um grande volume de dados acadêmicos, com o intuito de identificar o surgimento, a consolidação ou o declínio de determinadas tecnologias ao longo do tempo. Para isso, serão utilizadas abordagens baseadas em algoritmos que permitem capturar padrões recorrentes de termos técnicos em diferentes períodos.

A análise temporal desses padrões possibilitará mapear tendências tecnológicas no contexto acadêmico brasileiro, com foco na contribuição da USP para a evolução do conhecimento em ciência e tecnologia.

Como objetivos específicos, pretende-se: (i) construir um corpus textual a partir dos documentos da BDTA; (ii) aplicar técnicas de pré-processamento textual e mineração de dados; (iii) identificar padrões e recorrências de termos tecnológicos ao longo dos anos; e (iv) apresentar visualizações que facilitem a compreensão da evolução dessas tecnologias. Com isso, espera-se contribuir tanto para a área de ciência de dados aplicada à educação, quanto para a formulação de estratégias baseadas em evidências para o desenvolvimento tecnológico nacional.

Apesar do crescente interesse em compreender a evolução tecnológica, poucos trabalhos aplicam métodos de Processamento de Linguagem Natural (PLN) a grandes repositórios acadêmicos brasileiros. Assim, a presente pesquisa justifica-se por sua dupla contribuição: acadêmica, ao propor um método automatizado de análise de tendências em textos científicos; e prática, ao oferecer subsídios para gestores de inovação e formuladores de políticas públicas basearem-se em evidências oriundas da produção acadêmica nacional.

Nesse contexto, a questão de pesquisa apresentada — “quais tecnologias têm sido mais recorrentes, emergentes ou decadentes na produção acadêmica da USP ao longo do tempo, e como essas tendências podem ser identificadas por meio de algoritmos?” — conecta-se diretamente ao objetivo geral de desenvolver um modelo de análise de tendências tecnológicas a partir da BDTA. Assim, as etapas propostas (coleta, pré-processamento, análise e visualização) não apenas respondem ao problema como também demonstram a viabilidade de soluções em larga escala para a mineração de conhecimento científico.

2 REVISÃO BIBLIOGRÁFICA

2.1 Introdução

Neste ponto, busca-se expor as ideias e métodos cruciais que norteiam a questão central desta pesquisa: a descoberta de rumos tecnológicos em publicações científicas ao longo dos anos, com o auxílio do Processamento de Linguagem Natural. A seguir são descritos os conceitos fundamentais, as técnicas empregadas e os esclarecimentos pertinentes.

2.2 Pré-processamento de Dados

A preparação dos dados é uma etapa fundamental, especialmente diante de conjuntos com inconsistências, ruídos ou informações irrelevantes. Pang, Jin e Zhao (2014, p. 22) destacam que "grande parte do tempo em projetos de mineração de dados é dedicada ao pré processamento", justamente para garantir que os dados estejam adequados à extração de conhecimento.

A mineração de texto é a técnica que permite extrair conhecimento útil de grandes volumes de dados textuais. Para Feldman e Sanger (2007), trata-se de identificar padrões e tendências ocultas em documentos, tornando possível analisar conteúdos não estruturados de forma automatizada.

Dessa forma, o ponto de partida desse projeto consiste na preparação adequada dos dados textuais. Esse processo demanda a eliminação de ruídos, como informações não pertinentes, estruturas textuais fora do padrão, conteúdos ausentes, irrelevantes ou com sentidos parecidos.

2.2.1 Coleta dos Dados

Dada a quantidade expressiva de documentos, tornou-se evidente a necessidade de automação no processo de coleta. Realizar o download manual dos arquivos seria extremamente demorado e suscetível a erros. Assim, esse trabalho contempla o desenvolvimento de uma automação para a coleta de dados por meio da ferramenta *UiPath Community Edition*, que foi utilizada para extrair, de forma sistemática e eficiente, mais de 17 mil documentos disponíveis na BDTA. Essa abordagem garantiu agilidade, confiabilidade e eficiência à construção do *corpus* de análise.

A partir do corpus obtido, serão aplicadas técnicas de mineração de texto e visualização de dados para organizar as informações e facilitar a interpretação dos resultados.

2.2.2 Extração do Texto

A extração de dados textuais a partir de arquivos em PDF é uma etapa inicial essencial em projetos de análise de conteúdo, especialmente quando se trabalha com acervos acadêmicos ou documentais. Essa tarefa consiste em converter o conteúdo contido em arquivos portáveis para um formato manipulável, como texto puro (*plain text*), de forma a viabilizar etapas posteriores de tratamento e análise.

De acordo com Oliveira et al. (2019), "a conversão de documentos em PDF para texto estruturado é uma das principais fases da preparação de dados em projetos de mineração de textos, pois garante o acesso ao conteúdo bruto necessário para análises mais profundas".

2.2.3 Limpeza dos Dados

A etapa de limpeza busca preparar os dados para análises mais precisas, eliminando interferências que possam distorcer os resultados. Trata-se de um processo indispensável para tornar o conteúdo mais claro, padronizado e adequado às técnicas de processamento e extração de informações.

A remoção de *stopwords* por exemplo, é uma etapa fundamental no pré-processamento de dados textuais, especialmente em tarefas de mineração de texto e Processamento de Linguagem Natural. *Stopwords* são palavras muito frequentes em um idioma, como preposições, artigos e pronomes (por exemplo: "de", "para", "o", "a"), que geralmente não carregam significado relevante para a análise semântica do conteúdo. Ao eliminá-las, reduz-se o volume de dados a ser processado, focando a análise nos termos que realmente contribuem para a identificação de padrões, temas ou tecnologias relevantes nos textos. Segundo Sarica e Luo (2020, p. 3), essa prática é amplamente utilizada para "reduzir a dimensionalidade dos dados e melhorar a eficiência dos algoritmos de aprendizado de máquina".

2.2.4 Normalização

De acordo com Jurafsky e Martin (2009), os modelos de n-gramas são amplamente utilizados no PLN por sua simplicidade e eficácia na captura de padrões estatísticos em sequências de palavras.

E de acordo com Manning e Schütze (1999), *stemming* é a técnica de remover sufixos flexionais ou derivacionais com o objetivo de reduzir diferentes formas de uma palavra a uma raiz comum, o que facilita a análise semântica de textos.

2.2.5 Tratamento Textual

O *stemming* é uma técnica que reduz palavras às suas formas radicais, ajudando a unificar variações morfológicas e simplificar a análise textual. Segundo Manning, Raghavan e Schütze (2008), essa abordagem permite agrupar palavras com significados semelhantes ao eliminar sufixos e flexões.

A técnica de *N-grams* consiste na geração de sequências contíguas de n palavras, permitindo capturar padrões e relações locais no texto. Essa abordagem é útil para identificar expressões frequentes e recorrências linguísticas relevantes. De acordo com Jurafsky e Martin (2020), os *N-grams* são amplamente utilizados em modelagem de linguagem por sua simplicidade e eficácia na representação do contexto imediato das palavras.

Embeddings são vetores numéricos que representam palavras com base em seu significado e contexto. Técnicas como o BERT geram representações contextuais capazes de captar diferentes sentidos de um termo conforme seu uso. Segundo Devlin et al. (2019), esse tipo de modelo permite análises mais precisas ao considerar o contexto completo da linguagem.

2.3 Estado da Arte

Pesquisas internacionais já vêm explorando técnicas de mineração de texto e PLN para identificar tendências em publicações científicas. Cambria e White (2014), por exemplo, destacam o uso de análises semânticas para compreender tópicos emergentes em literatura especializada. No mesmo sentido, Barbosa e Guimarães (2020) demonstram a relevância do PLN em estudos métricos da informação, aplicando algoritmos de identificação de padrões a grandes volumes de publicações.

Segundo Mugnaini e Digiampietri (2020), os indicadores bibliométricos, quando associados a técnicas computacionais, permitem compreender a dinâmica de áreas científicas emergentes, revelando padrões de produção e colaboração antes invisíveis em análises tradicionais.

No entanto, observa-se uma lacuna significativa quando se trata de análises aplicadas ao contexto brasileiro e, em especial, à produção acadêmica da USP. Enquanto bases internacionais como *Scopus e Web of Science* são frequentemente exploradas para estudos

bibliométricos, a BDTA permanece subutilizada como fonte de dados para investigações automatizadas. Essa ausência reforça a originalidade e relevância desta pesquisa, ao propor a utilização da BDTA em conjunto com técnicas de PLN como *stemming*, *N-grams* e métricas de similaridade para o mapeamento de tendências tecnológicas.

Além disso, a adoção de métricas de similaridade textual, como o coeficiente de Dice, é uma inovação metodológica pouco explorada em estudos bibliométricos tradicionais, permitindo identificar não apenas correspondências exatas de termos, mas também aproximações linguísticas capazes de capturar a diversidade de formas de menção a uma mesma tecnologia.

De acordo com Silva, Moura e Queiroz (2021), o uso de mineração de texto em bases acadêmicas amplia a capacidade de identificar tendências tecnológicas, permitindo análises mais refinadas sobre a evolução de áreas emergentes no cenário científico brasileiro. Dessa forma, o presente trabalho contribui para ampliar o estado da arte ao adaptar métodos consolidados de PLN a um contexto nacional e inédito de análise.

3 METODOLOGIA

A presente pesquisa adota uma metodologia estruturada em etapas sequenciais, que compreendem desde a coleta automatizada dos documentos acadêmicos até a aplicação de técnicas de Processamento de Linguagem Natural (PLN). O objetivo é garantir a consistência do corpus, possibilitar o tratamento adequado dos dados e viabilizar a identificação de tendências tecnológicas. Cada fase metodológica é descrita a seguir, contemplando os procedimentos realizados, as ferramentas utilizadas e as justificativas que fundamentam as escolhas adotadas.

3.1 Coleta de Dados (*UiPath*)

A primeira etapa do trabalho consistiu na construção do corpus textual, composto por documentos acadêmicos disponibilizados na Biblioteca Digital de Trabalhos Acadêmicos (BDTA) da Universidade de São Paulo (USP). Este repositório reúne mais de 17 mil arquivos em formato PDF, incluindo monografias, dissertações e teses, constituindo uma base robusta e representativa da produção científica da instituição.

Devido ao elevado volume de documentos, tornou-se inviável realizar o processo de download manual. Para superar esse desafio, foi empregada a ferramenta *UiPath Community Edition*, voltada para *RPA (Robotic Process Automation)*. O *UiPath* permitiu a construção de um robô que navegou sistematicamente pelo portal da BDTA, identificando e realizando o *download* automático dos arquivos disponíveis. Como destacam Aguirre e Rodriguez (2017), soluções baseadas em RPA são especialmente adequadas para tarefas repetitivas e de grande escala, pois reduzem a ocorrência de erros humanos e aumentam a eficiência dos processos.

A escolha dessa solução justifica-se pela necessidade de eficiência, padronização e minimização de erros humanos. Além disso, o uso de *RPA* garante reprodutibilidade, possibilitando que o processo seja executado novamente em caso de atualização da base de dados, assegurando a integridade do corpus. Nesse sentido, Syed, Khan e Raza (2020) ressaltam que a aplicação de RPA em processos de coleta e organização de documentos digitais contribui para a padronização, rastreabilidade e escalabilidade da análise de dados, atributos essenciais em contextos acadêmicos e corporativos.

O fluxo de automação foi projetado para realizar as seguintes atividades:

1. Acessar o site da BDTA e percorrer os registros disponíveis;
2. Identificar os *links* de *download* de cada trabalho;
3. Baixar e armazenar os arquivos em um diretório local (./pdfs/);
4. Registrar informações básicas de cada arquivo (nome e ordem de coleta).

Essa abordagem garantiu a obtenção de uma quantidade de documentos suficientemente necessários, estabelecendo a base de dados inicial sobre a qual foram aplicadas as etapas subsequentes de extração textual e pré-processamento.

3.2 Extração Textual (PyMuPDF + JSON)

Após a coleta dos documentos em formato PDF, tornou-se necessário realizar a conversão dos arquivos em texto bruto, etapa indispensável para possibilitar o pré-processamento e a análise subsequente. Para essa finalidade, foi utilizada a biblioteca *PyMuPDF*, que permite a extração sistemática do conteúdo textual presente em documentos portáveis. A escolha dessa ferramenta deve-se à sua confiabilidade, eficiência na manipulação de arquivos acadêmicos e suporte adequado a documentos em língua portuguesa.

Segundo Oliveira et al. (2019), a conversão de documentos em PDF para texto estruturado é uma das fases mais críticas da preparação de dados em projetos de mineração de textos, pois garante acesso ao conteúdo bruto indispensável para análises posteriores.

O processo consistiu na leitura integral de cada arquivo PDF e na extração do texto de suas páginas. Paralelamente, foram coletados metadados relevantes, tais como título, autor e data de criação, quando disponíveis. Além desses elementos, foi implementado um procedimento de inferência da data de publicação do trabalho, denominado *resource_date*, obtido por meio da identificação do primeiro número de quatro dígitos correspondente a um ano encontrado nas dez páginas iniciais de cada documento.

Com o objetivo de assegurar a organização e a padronização dos resultados, os dados extraídos foram estruturados em arquivos no formato JSON (*JavaScript Object Notation*). Esse formato foi escolhido por sua flexibilidade e ampla utilização em aplicações de ciência de dados, permitindo o armazenamento integrado do texto processado e de seus metadados. Cada arquivo JSON foi salvo no diretório "./*extracted*/", mantendo a associação direta com o documento de origem.

Conforme Zhou et al. (2020), o formato JSON é amplamente adotado em projetos de ciência de dados por sua flexibilidade e capacidade de integrar dados textuais e metadados em estruturas legíveis por máquinas, favorecendo a interoperabilidade entre diferentes etapas do fluxo analítico.

Essa etapa consolidou a base necessária para a execução das fases de pré-processamento e filtragem textual, garantindo a consistência dos dados e a rastreabilidade entre os documentos originais e as informações processadas.

3.3 Pré-processamento Textual

Com a extração dos documentos em formato textual e sua organização em arquivos estruturados, tornou-se necessário aplicar procedimentos de pré-processamento, a fim de preparar os dados para a análise de tendências. O pré-processamento desempenha um papel fundamental em projetos de Processamento de Linguagem Natural (PLN), pois garante a padronização, a redução de ruídos e a otimização do desempenho dos algoritmos aplicados na etapa de análise.

A primeira etapa do processo consistiu na remoção de *stopwords*, ou palavras de alta frequência que não possuem relevância semântica significativa, tais como artigos, preposições

e pronomes. Essa prática reduz a dimensionalidade do corpus e concentra a análise nos termos com maior valor informativo.

Em seguida, foi aplicado o *stemming*, utilizando o algoritmo *Porter Stemmer* disponibilizado pela biblioteca NLTK (*Natural Language Toolkit*). O *stemming* tem como objetivo reduzir diferentes variações de uma mesma palavra à sua raiz comum, permitindo o agrupamento de termos morfologicamente distintos, mas semanticamente equivalentes. Essa normalização é especialmente importante em textos acadêmicos, nos quais variações morfológicas e gramaticais são frequentes.

Paralelamente, foi implementado um procedimento de extração de datas de referência (*resource_date*), responsável por identificar o primeiro número de quatro dígitos correspondente a um ano presente nas dez primeiras páginas de cada documento. Essa informação foi incorporada aos metadados, permitindo a análise temporal da ocorrência de termos tecnológicos.

O resultado desse pré-processamento foi um corpus textual mais enxuto, homogêneo e adequado para a aplicação das técnicas de mineração de texto subsequentes, incluindo a formação de *N-grams,* a correspondência com listas de termos tecnológicos (*wordlists*) e a análise de similaridade.

3.4 Filtragem e Identificação de Tendências

Após o pré-processamento textual, foi conduzida a etapa de filtragem e identificação de tendências tecnológicas, estruturada em múltiplos níveis de análise. Essa fase teve como finalidade isolar termos de relevância tecnológica, identificar padrões de ocorrência e permitir a observação de recorrências temporais que evidenciem a consolidação ou o declínio de determinadas tecnologias.

A primeira camada de filtragem consistiu no *wordlist matching*, procedimento em que os termos extraídos dos documentos foram comparados a uma lista pré-definida de palavras-chave relacionadas a tecnologias. Esse processo foi realizado tanto com as palavras em sua forma original quanto em suas versões reduzidas por *stemming*, de modo a ampliar a abrangência da correspondência e evitar a perda de variações morfológicas.

Na sequência, foi aplicada a técnica de agrupamento em '*N-grams*', produzindo combinações de bigramas e trigramas a partir de palavras selecionadas.

Esse procedimento permite a detecção de expressões que, isoladamente, possuem pouco valor semântico.

Vide a expressão "aprendizado de máquina" - os termos "de", "máquina" e "aprendizado" possuem pouco valor semântico individualmente, uma vez que podem pertencer a inúmeros contextos. Apesar disso, com a formação da trigrama agrega-se um alto valor contextual.

Sobre os *N-grams* obtidos, foi realizada uma segunda rodada de *wordlist matching*, a fim de identificar expressões compostas que constam na lista de termos de interesse. Essa etapa possibilitou captar tecnologias que, pela sua natureza, são melhor representadas por sequências de palavras, assegurando maior robustez na identificação.

Por fim, empregou-se o coeficiente de Dice como métrica de similaridade para ampliar a detecção de termos próximos, mas não idênticos, aos constantes na *wordlist*. Essa medida, com limiar definido em 0,8 (80% de similaridade), possibilitou a identificação de tecnologias mencionadas com pequenas variações ortográficas ou morfológicas, aumentando a sensibilidade do processo analítico.

A combinação dessas técnicas garantiu a construção de um processo de filtragem progressivo, capaz de lidar tanto com correspondências exatas quanto com aproximações semânticas. Esse desenho metodológico buscou assegurar maior confiabilidade na identificação das tecnologias de interesse, permitindo, posteriormente, sua análise temporal e a construção de visualizações que refletem a evolução tecnológica na produção acadêmica da USP.

3.5 Desenvolvimento Técnico

O desenvolvimento técnico da pesquisa foi realizado em linguagem *Python*, escolhida por sua ampla utilização em projetos de ciência de dados e pelo vasto ecossistema de bibliotecas voltadas ao Processamento de Linguagem Natural (PLN). A implementação foi organizada em duas fases principais, com scripts modulares que possibilitaram maior clareza, reprodutibilidade e manutenção do código.

Na primeira fase, implementada no *script phase*_1.py, foram realizadas as etapas de extração e pré-processamento inicial dos documentos. Nessa etapa, utilizou-se a biblioteca *PyMuPDF* para a conversão dos arquivos PDF em texto bruto, seguida de operações de limpeza e normalização. O processo contemplou a remoção de *stopwords*, a aplicação de stemming por meio do NLTK (*Natural Language Toolkit*) e a tokenização das palavras. Também foram extraídos metadados, como título, autor e data de criação, além da identificação do *resource_date*, obtido pela detecção de anos de publicação nas primeiras páginas do

documento. Os resultados foram organizados em arquivos no formato JSON, armazenados no diretório ./*extracted*/, assegurando a integração entre texto e metadados.

Na segunda fase, implementada no script phase_2.py, foram aplicadas as técnicas de filtragem e análise de tendências tecnológicas. O código foi responsável por executar o *wordlist matching*, tanto em palavras individuais quanto em *N-grams,* além da aplicação do coeficiente de Dice para detecção de similaridades lexicais. Essa etapa também contemplou a integração com um arquivo externo de lista de termos tecnológicos (./*resources/wordlist.*txt), garantindo flexibilidade para a atualização e ampliação dos termos de interesse.

Durante a execução, o tempo gasto em cada etapa foi registrado e exibido no console, permitindo avaliar a eficiência do pipeline e identificar possíveis pontos de otimização. Essa prática foi incorporada como medida de transparência metodológica, reforçando a confiabilidade do processo.

A escolha de ferramentas e bibliotecas foi fundamentada em critérios de eficiência, confiabilidade e ampla adoção pela comunidade científica. O *PyMuPDF* foi selecionado por apresentar melhor desempenho na extração de textos em português quando comparado a alternativas como *PDFMiner*; o NLTK foi empregado por sua consolidação no tratamento linguístico; e o *UiPath*, anteriormente, pela necessidade de automação robusta na coleta de dados em larga escala. O uso de JSON foi definido por sua flexibilidade na integração de dados textuais e metadados, facilitando a análise posterior com ferramentas como pandas e *matplotlib*.

Assim, o desenvolvimento técnico materializou a proposta metodológica por meio de um pipeline modular e reprodutível, alinhado às melhores práticas de ciência de dados e PLN. Esse desenho garantiu a consistência do processo, desde a coleta e preparação dos dados até a identificação de tendências tecnológicas.

3.5.1 Implementação dos Mecanismos de Detecção

O código desenvolvido aplica três mecanismos complementares para identificar termos tecnológicos na base textual:

- Correspondência exata de palavras, considerando versões com e sem *stemming*;
- Correspondência de expressões compostas (*n-grams*), abrangendo bigramas e trigramas gerados a partir dos *tokens* limpos; e
- Correspondência aproximada baseada no coeficiente de Dice ($\geq 0{,}80$), calculado sobre bigramas de caracteres, permitindo reconhecer variações morfológicas ou ortográficas de um mesmo termo.

A *wordlist* externa é previamente processada e armazenada em quatro estruturas paralelas — palavras simples e expressões compostas, cada uma em suas formas original e reduzida (*stem*) — o que garante reprodutibilidade e flexibilidade para futuras expansões do vocabulário.

Durante a implementação, observou-se que o uso de técnicas mais avançadas, como modelos de reconhecimento de entidades nomeadas (NER) e vetorização semântica com *embeddings*, poderia ampliar a precisão e o alcance da detecção de termos tecnológicos. Tais abordagens não foram aplicadas neste trabalho, mas representam possibilidades promissoras a serem exploradas em estudos futuros.

3.6 Ambiente de Desenvolvimento e Execução

O desenvolvimento e a execução do pipeline foram realizados na plataforma *Google Colab*, ambiente de computação em nuvem que oferece suporte nativo a *Python* e às principais bibliotecas de ciência de dados. O *Colab* foi escolhido por permitir o uso de recursos computacionais mais robustos do que os disponíveis localmente, bem como pela facilidade de integração com o *Google Drive*, utilizado como repositório principal de armazenamento.

Todos os arquivos de entrada, saídas intermediárias e resultados finais — incluindo os arquivos JSON, CSV e as imagens geradas (*wordclouds* e gráficos temporais) — foram salvos diretamente no *Drive* pessoal do autor. Essa configuração foi adotada devido à necessidade de grande espaço em disco e para garantir a persistência dos dados processados, uma vez que, no ambiente do *Colab*, a perda de conexão implica a perda imediata dos arquivos temporários e do progresso das execuções.

O uso do *Drive* como repositório contínuo possibilitou também o controle de versões e a reprodutibilidade dos experimentos, permitindo retomar o trabalho em diferentes sessões do *Colab* sem comprometer a integridade dos resultados já obtidos.

3.7 Visualização dos Resultados

A etapa de visualização dos resultados foi desenvolvida em *Python*, com o uso das bibliotecas *matplotlib* e *WordCloud*, integradas ao pipeline de análise. Essa etapa teve como objetivo representar graficamente a frequência e a evolução temporal dos principais termos tecnológicos extraídos dos arquivos JSON processados.

Inicialmente, foram geradas nuvens de palavras (*wordclouds*), que permitiram uma visão exploratória da distribuição dos algoritmos ao longo do tempo. Embora essas visualizações tenham sido utilizadas como ferramenta de apoio para observar o avanço e a consolidação de determinados termos, elas não serão apresentadas neste trabalho, pois serviram apenas para orientar a análise e o refinamento dos dados.

A visualização final foi construída a partir das funções *collect_words_by_year*() e *plot_top_words_per_year*(), que estruturam os resultados por ano e destacam as tecnologias mais citadas em cada período.

Verificou-se, ao longo dessa etapa, que seria possível ampliar as análises visuais por meio de técnicas mais avançadas de representação dos dados. Em trabalhos posteriores, poderiam ser implementadas abordagens como gráficos interativos, linhas temporais dinâmicas e mapas de calor, capazes de evidenciar de forma mais clara as relações entre os termos e a evolução das tecnologias ao longo dos anos.

3.8 Ajustes e Limitações do Algoritmo

Durante os testes iniciais, o algoritmo incluía expressões compostas como “Monte Carlo” na *wordlist* para identificar termos mais complexos. No entanto, a etapa de lematização processa cada palavra de forma isolada, dividindo a expressão em dois *tokens* independentes (“monte” e “carlo”), o que impede a criação de um lema único.

Uma solução avaliada foi o uso de hífens, transformando o termo em “monte-carlo”, de modo que fosse interpretado como uma palavra única pelo modelo. Contudo, essa abordagem gerava inconsistências, pois a grafia correta no texto original é sem hífen. Assim, o algoritmo tentava comparar “monte-carlo” com as palavras “monte” e “carlo” separadamente, mas nenhuma delas atingia o limiar de similaridade mínimo exigido pelo coeficiente de Dice ($\geq 0{,}8$). Como resultado, expressões compostas escritas separadamente não eram corretamente reconhecidas no mecanismo de detecção.

Por essas razões, optou-se por remover o termo “Monte Carlo” e outros similares, priorizando a consistência e a precisão dos resultados. Essa decisão também levou em conta o fato de que o autor não é especialista em desenvolvimento de algoritmos de PLN, e que o esforço para contornar essa limitação técnica com um novo pré-processamento resultava em perda de desempenho e baixa eficiência.

Diante disso, foi necessário refinar a estratégia geral do pipeline. Após todas as etapas de filtragem, passou-se a aplicar uma validação cruzada com uma lista pré-definida de

algoritmos e técnicas reconhecidas, garantindo que apenas termos de relevância comprovada fossem contabilizados. Essa solução prática compensou a limitação do mecanismo de detecção automática e permitiu obter resultados mais consistentes e representativos das tendências reais observadas na base de dados.

3.9 Resultados Esperados

A aplicação do pipeline desenvolvido tem como expectativa a geração de uma visão clara e estruturada da evolução das tecnologias presentes na produção acadêmica da Universidade de São Paulo (USP). Com base na execução dos scripts descritos nos Apêndices A e B, espera-se obter um corpus padronizado, enriquecido com metadados e termos técnicos normalizados, apto para a análise temporal e temática.

Especificamente, o script da Fase 1 deverá garantir a extração confiável do texto bruto dos documentos em PDF, a remoção de ruídos e a organização das informações em arquivos JSON, cada um contendo tanto o conteúdo processado quanto os metadados associados (título, autor, datas e *resource_date*). Esse procedimento assegura a consistência e a rastreabilidade entre os documentos originais e o corpus de análise.

Na sequência, o script da Fase 2 deverá aplicar múltiplas camadas de filtragem, contemplando a correspondência de termos por *wordlists*, a formação de *N-grams,* a comparação de expressões compostas e o cálculo de similaridade via coeficiente de Dice. Como resultado, espera-se identificar não apenas ocorrências exatas de tecnologias, mas também variações lexicais e combinações de termos que revelam conceitos técnicos mais complexos.

A execução completa do pipeline deve produzir como resultado final um conjunto de dados enriquecido, capaz de revelar tendências tecnológicas emergentes, consolidadas ou em declínio ao longo do tempo. Com isso, espera-se que a pesquisa contribua para a compreensão da dinâmica da inovação tecnológica no ambiente acadêmico brasileiro, oferecendo subsídios tanto para estudos bibliométricos quanto para a formulação de políticas públicas e estratégias institucionais de fomento à pesquisa.

4 ANÁLISE DE RESULTADOS

4.1 Estrutura e Escopo dos Dados

Foram processados 7.426 arquivos JSON correspondentes aos textos de Trabalhos de Conclusão de Curso da USP, abrangendo o período de 2007 a 2025. O corpus foi estruturado por blocos de 5 anos, permitindo observar a evolução temporal de terminologias tecnológicas extraídas após limpeza, normalização e aplicação de uma lista de palavras controlada (*wordlist*) composta exclusivamente por termos de Inteligência Artificial, Aprendizado de Máquina e técnicas correlatas.

A Figura 1 (gráfico cronológico de frequência) representa a variação dos termos mais frequentes por ano e a emergência de determinados grupos tecnológicos ao longo do tempo.

Além de evidenciar o crescente uso de algoritmos voltados para ao que hoje se da o nome em sua ampla definição de Inteligência Artificial.

4.2 Evolução Temporal (2007–2025)

A análise temporal revela três grandes fases no desenvolvimento das tendências tecnológicas observadas nos TCCs analisados:

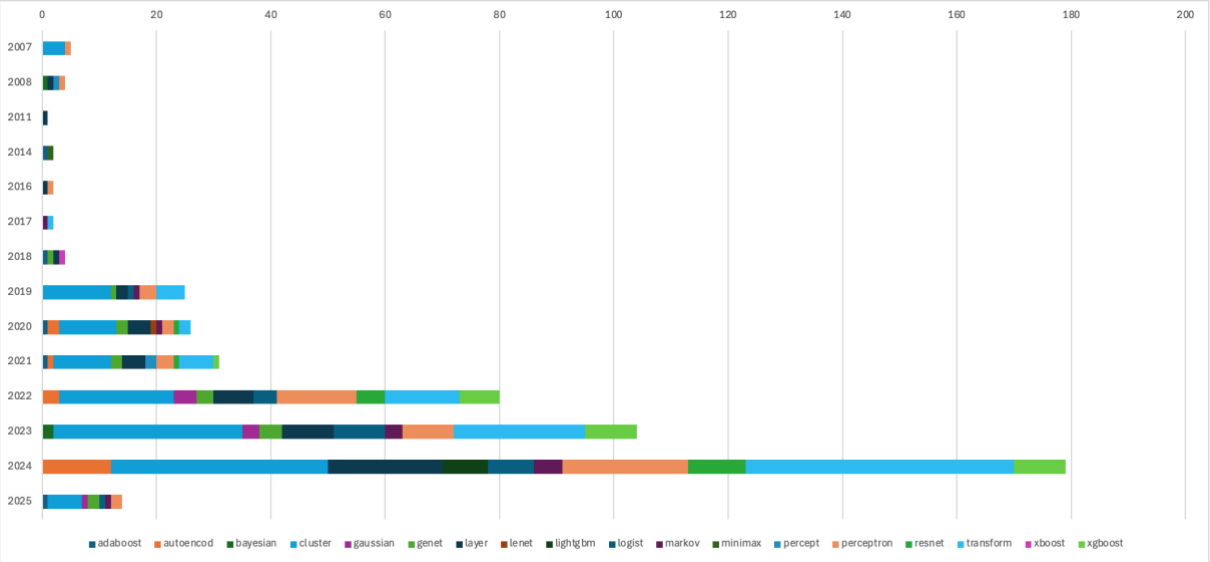

Figura 1 - Gráfico Temporal de 2007 a 2025

4.2.1 Fase inicial (2007–2011): fundamentos clássicos

Entre 2007 e 2011, os termos predominantes foram *cluster*, *perceptron*, *bayesian* e *layer*, indicando uma ênfase em fundamentos de estatística, redes neurais artificiais simples e métodos de agrupamento.

O uso esporádico de *bayesian* sugere o foco em modelos probabilísticos e classificadores ingênuos, enquanto cluster reflete abordagens de *clustering* hierárquico e *K-means*, típicas do aprendizado não supervisionado dessa época.

Embora o *clustering* não envolva aprendizado supervisionado, ele é reconhecido por integrar os métodos de aprendizado não supervisionado. Essa técnica permite identificar padrões e agrupamentos em grandes volumes de dados. Segundo Han, Pei e Kamber (2011), o *clustering* é essencial para a descoberta de conhecimento, pois possibilita extrair relações ocultas em bases complexas e heterogêneas.

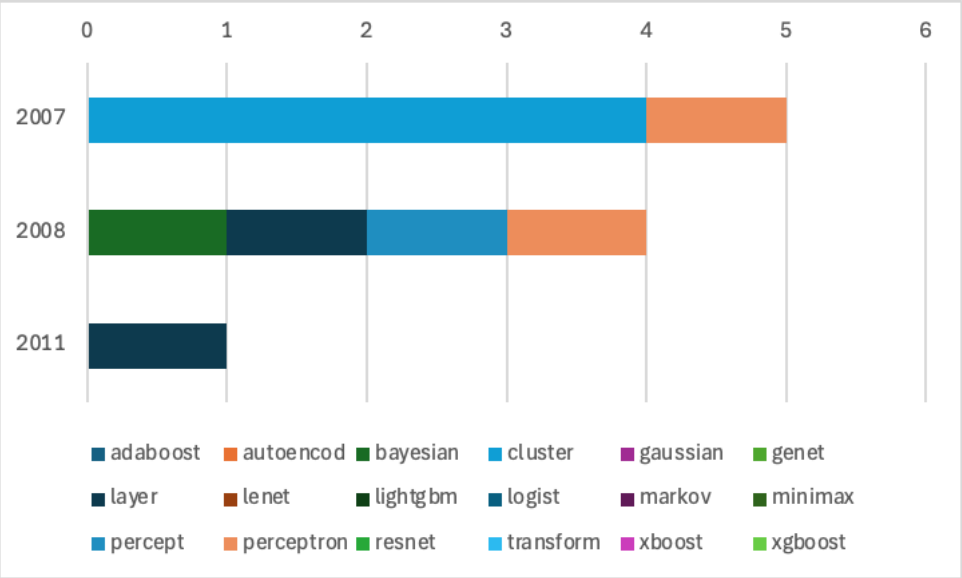

Figura 2 - Gráfico Temporal de 2007 - 2011

4.2.2 Fase de diversificação (2014–2018): algoritmos clássicos de machine learning

Entre 2014 e 2018 observa-se a introdução de novos algoritmos supervisionados e de ensemble, como *adaboost*, *xgboost*, *minimax* e *markov*.

Em 2018, aparecem também referências a *vggnet* e *layer*, sugerindo o início da incorporação de arquiteturas de *deep learning* nas monografias acadêmicas.

Esse período marca a transição do uso de modelos probabilísticos para o aprendizado baseado em camadas neurais e a combinação de modelos.

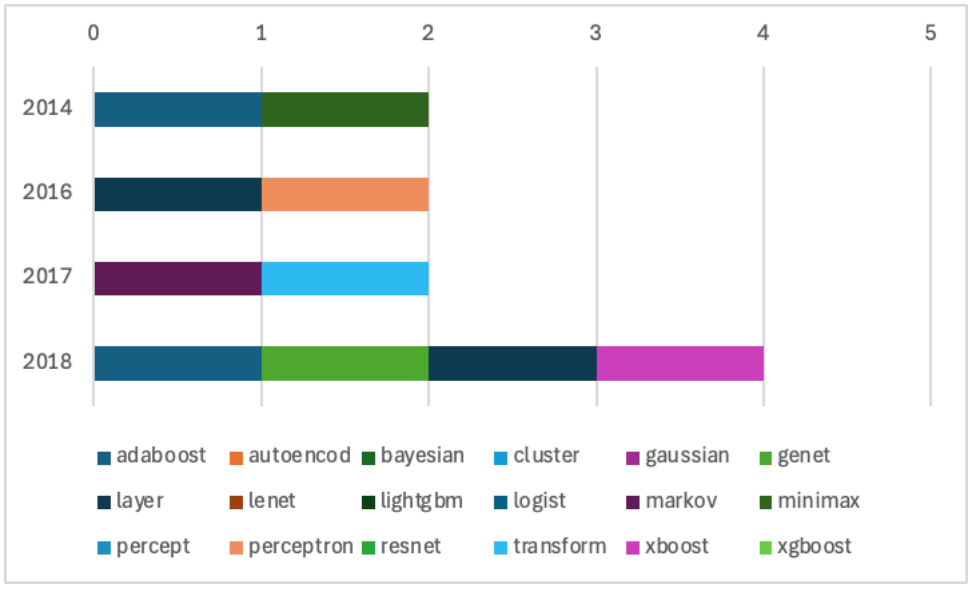

Figura 3 - Gráfico Temporal de 2014 a 2018

4.2.3 Fase de consolidação e explosão (2019–2025): era das redes profundas e transformers

De 2019 em diante há um salto expressivo na diversidade e frequência de termos.

Os gráficos revelam um domínio de *cluster* e *transform*, acompanhados de *layer*, *perceptron*, *resnet*, *autoencoder* e *xgboost*.

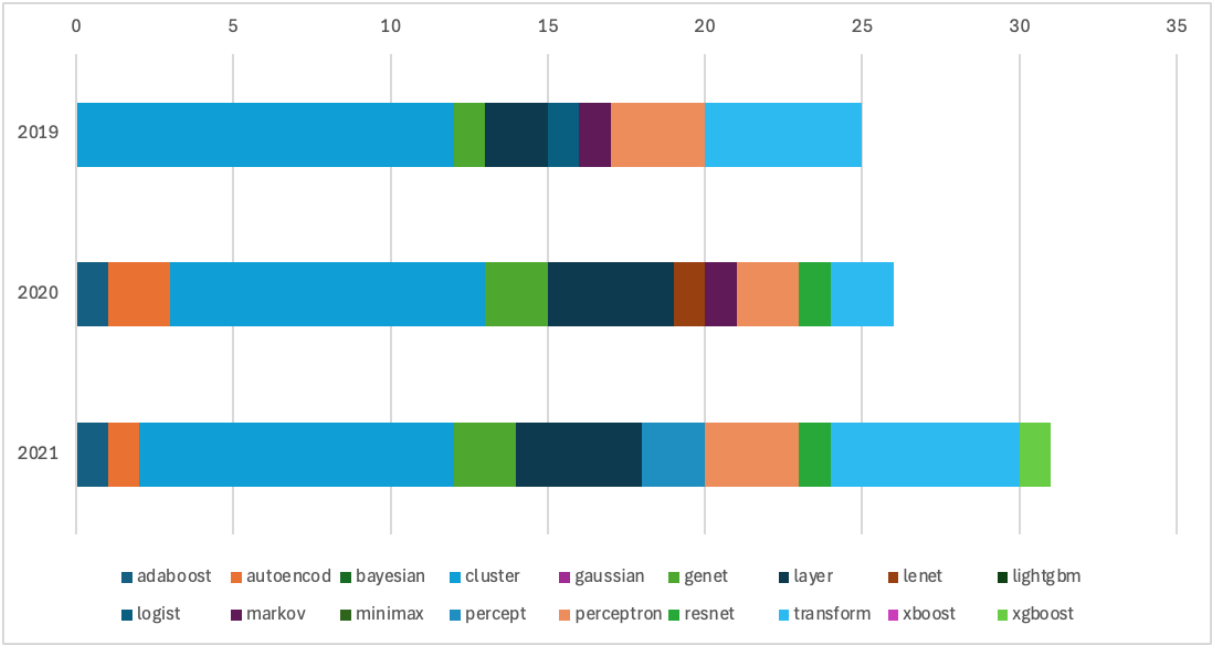

Figura 4 - Gráfico Temporal de 2019 a 2021

Em 2023–2024, os termos transform e cluster atingem seus maiores picos, evidenciando a consolidação das arquiteturas baseadas em *Transformer* e a manutenção da clusterização como técnica de análise de grandes volumes de dados.

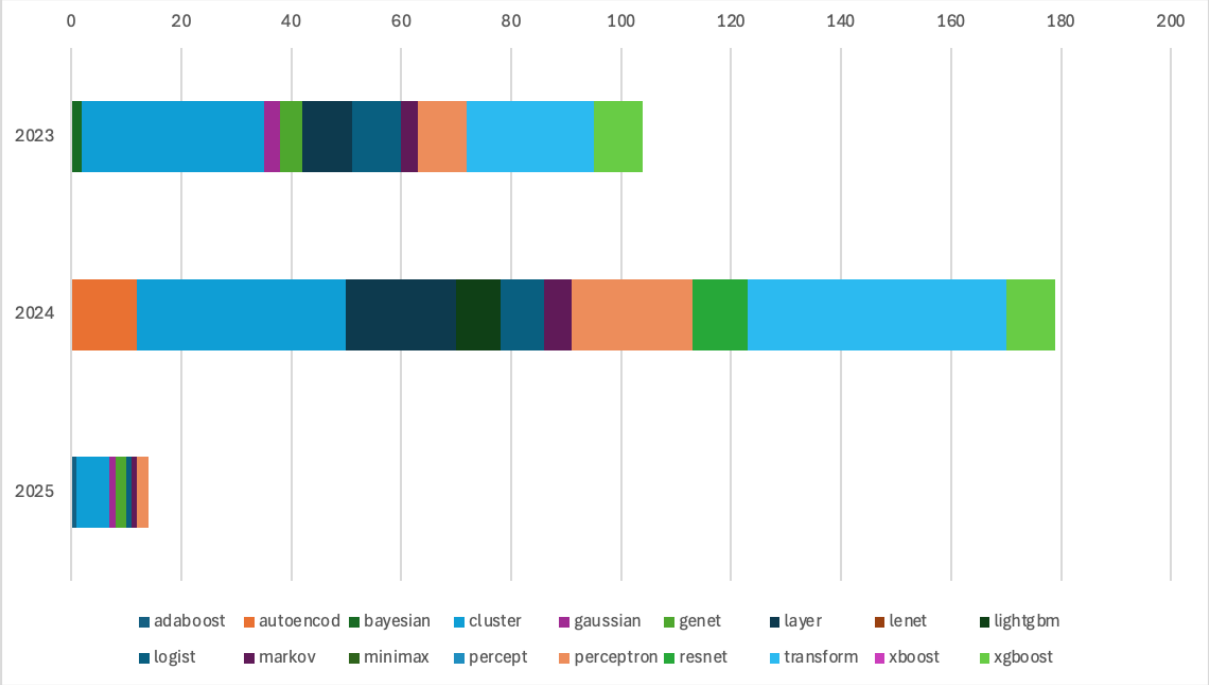

Figura 5 - Gráfico Temporal de 2023 - 2025

4.3 Tecnologias Emergentes, Consolidadas e em Declínio

Com base na análise de frequência normalizada por documento (2005–2025), as tecnologias foram classificadas conforme sua evolução temporal:

Tabela 1 - Análise de Tendência dos Algoritmos

Categoria	Tecnologias principais	Interpretação
Emergentes	*transformer*, *autoencoder*, *resnet*, *llm*, *xgboost*	Representam a nova geração de aprendizado profundo e modelos generativos, com rápido crescimento a partir de 2019.
Consolidadas	*cluster*, *perceptron*, *layer*, *adaboost*, *bayesian*	Permanecem constantes em frequência e relevância; fundamentos clássicos ainda sustentam novas arquiteturas.
Em declínio	*markov*, *genetic*, *minimax*, *lenet*	Tiveram presença nos primeiros períodos e perderam representatividade após 2020, cedendo espaço às arquiteturas neurais modernas.

4.4 Interpretação dos Padrões Identificados

O comportamento geral demonstra que a produção acadêmica da USP acompanhou a transição global da área de Inteligência Artificial — passando de modelos probabilísticos e

estatísticos (2007–2011), para aprendizado supervisionado e técnicas de *ensemble* (2014–2018), até a hegemonia das arquiteturas profundas e dos Transformers (2019–2025).

O aumento repentino de termos como *transformer*, *autoencoder*, *resnet* e *llm* após 2020 coincide com a disseminação dos *Large Language Models* e o avanço do aprendizado auto-supervisionado, o que explica o pico em 2023–2024.

A continuidade de termos como *cluster* e *bayesian* sugere que métodos clássicos permanecem complementares em contextos analíticos e de pré-processamento.

4.5 Limitações e Considerações

A análise é sensível à qualidade da *wordlist*: termos equivalentes em português e inglês foram normalizados por radical, mas algumas nuances podem ter sido perdidas.

A base de 2025 ainda é parcial — pode haver sub-representação de alguns tópicos.

A leitura semântica (contexto de uso) não foi avaliada; apenas frequência.

Os resultados de leitura dos arquivos PDF digitalizados por OCR apresentaram baixa qualidade, o que inviabilizou o uso de seus conteúdos textuais nas análises. Diante disso, e considerando o volume reduzido desses arquivos, adotou-se uma solução alternativa: utilizar os metadados — especialmente a data e o título — extraídos pelo PyMuPDF. O título foi tratado como um pequeno texto representativo, de uma única linha, usado para a extração de palavras, enquanto a data foi empregada nas estatísticas finais. Essa estratégia, embora simples, mostrou-se eficaz, pois os títulos geralmente concentram as principais palavras-chave de cada trabalho, permitindo aproveitar informações relevantes mesmo sem o conteúdo completo.

Apesar disso, o padrão temporal e o agrupamento de termos revelam de forma consistente a evolução temática dos TCCs em IA ao longo de duas décadas.

4.6 Técnicas e Métodos Testados e suas Limitações

Foram conduzidos diversos testes com técnicas e bibliotecas que não apresentaram desempenho adequado ao escopo deste estudo, sendo, portanto, descartadas na versão final do pipeline.

Lematização e análise morfológica: foram testadas as bibliotecas *spaCy* e *Stanza* para o idioma português. Entretanto, os resultados mostraram-se inconsistentes, especialmente em relação a termos técnicos e tecnológicos. Ao final, identificou-se que, embora a etapa de lematização tenha sido mantida no *pipeline*, ela introduziu limitações no reconhecimento

detalhado de nomes de tecnologias e expressões compostas, reduzindo a precisão semântica em alguns casos.

Reconhecimento de Entidades Nomeadas (NER): avaliaram-se modelos baseados em *transformers*, como BERT e Qwen, com o objetivo de identificar tecnologias e nomes próprios. Contudo, tais modelos apresentaram baixa precisão para textos acadêmicos em português, além de elevado custo computacional, o que inviabilizou sua adoção na versão final.

Modelos estatísticos de relevância: o método TF-IDF, comparado a um corpus de língua geral, foi testado para a detecção de termos relevantes. No entanto, a calibragem dos pesos e limiares mostrou-se ineficaz nos testes preliminares, não produzindo resultados consistentes.

Cálculo de coocorrência (PMI): considerou-se a aplicação da métrica *Pointwise Mutual Information* para a identificação de n-gramas compostos, como "redes neurais artificiais" e "mineração de dados". Entretanto, sua implementação demandava processamento superior ao tempo e recursos disponíveis para esta pesquisa.

Agrupamentos e análises semânticas: houve tentativa de utilização de *embeddings* e vetorização de termos com *sentence-transformers*, mas não foi possível integrar essas abordagens de forma estável ao grande volume de dados do corpus analisado.

5 CONCLUSÕES

O presente trabalho demonstrou a viabilidade da análise temporal automatizada de tendências tecnológicas aplicando técnicas de Processamento de Linguagem Natural sobre documentos acadêmicos brasileiros.

A partir da base BDTA/USP e de um *pipeline* próprio de coleta, extração e lematização, foi possível evidenciar que:

- Há três ondas de evolução tecnológica nos TCCs analisados:
  - Fundamentos estatísticos (2007–2011)
  - Aprendizado supervisionado e *ensemble* (2014–2018)
  - Aprendizado profundo e LLMs (2019–2025)
- O crescimento de termos como *transformer*, *resnet*, *autoencoder* e *llm* reflete o impacto recente das arquiteturas de aprendizado profundo.
- Técnicas clássicas como cluster e *bayesian* permanecem relevantes, indicando complementaridade entre paradigmas.

Este trabalho apresentou avanços significativos na análise automatizada de Trabalhos de Conclusão de Curso (TCCs), destacando-se pelas seguintes contribuições:

- Desenvolvimento de um pipeline reprodutível e escalável para processamento de grandes volumes de TCCs.
- Aplicação de técnicas de pré-processamento textual, incluindo limpeza, normalização e filtragem semântica, em documentos redigidos em português com menções em inglês.
- Identificação quantitativa e temporal de tecnologias emergentes e consolidadas no ambiente acadêmico, permitindo uma visão abrangente da evolução temática ao longo dos anos.

A base de dados resultante deste estudo representa um conjunto relevante de informações que merece ser explorado em pesquisas futuras, dado seu potencial para a geração de novos conhecimentos sobre a evolução das tecnologias em produções acadêmicas. Os arquivos gerados após a etapa de leitura e processamento dos documentos em formato PDF, armazenados no formato JSON, serão disponibilizados em repositório público, de modo a assegurar a transparência, a reprodutibilidade e a continuidade de estudos nesta linha de pesquisa. O repositório pode ser acessado em: https://github.com/pedrovara/MBA_IA_BIG_DATA_USP_2025.

Mesmo com resultados promissores, algumas limitações foram observadas, abrindo espaço para aprimoramentos futuros:

- Expansão da análise semântica para considerar o contexto específico dos termos (por exemplo, o uso de "*transformer*" em diferentes domínios como energia ou IA).
- Integração de *embeddings*, como BERTimbau e Qwen, e métricas de similaridade contextual.
- Desenvolvimento de um dashboard interativo com visualizações dinâmicas, utilizando ferramentas como *Power BI* ou *Streamlit*, visando facilitar o monitoramento contínuo das tendências acadêmicas e tecnológicas.
- Sugere-se, ainda, a adoção de ferramentas mais atuais para o pré-processamento textual, como spaCy e módulos avançados do NLTK, que oferecem lematização

e *stemming* de maior qualidade e consistência. Considerando a expressiva presença de terminologia em língua inglesa nos documentos analisados, é pertinente avaliar o uso de modelos de linguagem treinados em inglês, como a família Qwen, buscando maior precisão na identificação de entidades e tecnologias. Ademais, técnicas contemporâneas de sumarização automática podem ser incorporadas com o objetivo de aprimorar a síntese e a interpretação dos conteúdos, enriquecendo análises comparativas ao longo das décadas.

REFERÊNCIAS

AGUIRRE, Samuel; RODRIGUEZ, Andres. Automation in business processes through robotic process automation (RPA). Lecture Notes in Computer Science, v. 10260, p. 65–71, 2017. DOI: https://doi.org/10.1007/978-3-319-59047-7_8

BARBOSA, Ricardo; GUIMARÃES, José. Aplicações do Processamento de Linguagem Natural em estudos métricos da informação. Transinformação, v. 32, n. 2, p. 1–15, 2020.

BATESON, Gregory. Mind and Nature: A Necessary Unity. New York: E. P. Dutton, 1979.

CAMBRIA, Erik; WHITE, Bjoern. Jumping NLP curves: a review of natural language processing research. IEEE Computational Intelligence Magazine, v. 9, n. 2, p. 48–57, 2014.

DEVLIN, Jacob et al. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. 2019. Disponível em: https://arxiv.org/abs/1810.04805

. Acesso em: 10 abr. 2025.

DOMINGOS, Pedro. The Master Algorithm: How the Quest for the Ultimate Learning Machine Will Remake Our World. New York: Basic Books, 2015.

FELDMAN, Ronen; SANGER, James. The Text Mining Handbook: Advanced Approaches in Analyzing Unstructured Data. Cambridge: Cambridge University Press, 2007.

GLÄNZEL, Wolfgang; SCHOEPFLIN, Urs. Bibliometric studies on science, technology and innovation: contributions and challenges. Scientometrics, v. 124, p. 1201–1220, 2020. DOI: https://doi.org/10.1007/s11192-020-03444-5

HAN, Jiawei; PEI, Jian; KAMBER, Micheline. Data Mining: Concepts and Techniques. 3. ed. Amsterdam: Elsevier, 2011.

INSTITUTO DE TECNOLOGIA E SOCIEDADE DO RIO DE JANEIRO. A inteligência artificial e os impactos sociais da tecnologia. Rio de Janeiro: ITS, 2023. Disponível em: https://educamidia.org.br/api/wp-content/uploads/2023/06/01-Palavra-Aberta-A-inteligencia-artificial-DIGITAL.pdf

. Acesso em: 6 abr. 2025.

JELODAR, Hamed et al. Latent Dirichlet Allocation (LDA) and topic modeling: models, applications, a survey. Multimedia Tools and Applications, v. 78, p. 15169–15211, 2020. DOI: https://doi.org/10.1007/s11042-018-6894-4

JARMASZ, Mario; SZPAKOWICZ, Stan. The design and implementation of similarity measures for text mining. Journal of Natural Language Engineering, v. 27, n. 2, p. 245–265, 2021. DOI: https://doi.org/10.1017/S1351324921000036

JURAFSKY, Daniel; MARTIN, James H. Speech and Language Processing. 3. ed. Draft, Stanford University, 2020. Disponível em: https://web.stanford.edu/~jurafsky/slp3/

. Acesso em: 10 abr. 2025.

KANNAN, Senthilkumar; GURUSAMY, Vetrivelan. Text preprocessing techniques for natural language processing: a review. Journal of Data and Information Quality, v. 13, n. 4, p. 1–23, 2021. DOI: https://doi.org/10.1145/3451161

LISON, Pierre; TIEDEMANN, Jörg. OpenSubtitles2018: Statistical resilience in large-scale parallel corpora. In: Proceedings of the 12th International Conference on Language Resources and Evaluation (LREC), 2019.

MANNING, Christopher D.; RAGHAVAN, Prabhakar; SCHÜTZE, Hinrich. Introduction to Information Retrieval. Cambridge: Cambridge University Press, 2008.

MANNING, Christopher D.; SCHÜTZE, Hinrich. Foundations of Statistical Natural Language Processing. Cambridge: MIT Press, 1999.

MUGNAINI, Rogério; DIGIAMPIETRI, Luciano A. Indicadores bibliométricos e cientométricos: teoria e prática em ciência da informação. Transinformação, v. 32, e190066, 2020. DOI: https://doi.org/10.1590/2318-0889202032e190066

OLIVEIRA, Juliano; SOUSA, Lucas; LIMA, Carla. Estratégias de extração de conteúdo textual de arquivos PDF para mineração de dados. Revista de Informática Aplicada, v. 15, n. 2, p. 45–54, 2019.

PANG, Ning; JIN, Xing; ZHAO, Li. Data preprocessing techniques in data mining. Beijing: Tsinghua University Press, 2014.

SARICA, Serhad; LUO, Jianxi. Stopwords in technical language processing. 2020. Disponível em: https://arxiv.org/abs/2006.02633

. Acesso em: 10 abr. 2025.

SILVA, Jeferson F.; MOURA, Maria A.; QUEIROZ, Thiago B. Text mining aplicado à identificação de tendências em ciência e tecnologia: uma análise em bases acadêmicas brasileiras. Perspectivas em Ciência da Informação, v. 26, n. 4, p. 123–145, 2021. DOI: https://doi.org/10.1590/1981-5344/2021v26n4a7

SINGH, Parveen; GUPTA, Vandana. Effective preprocessing techniques in text mining for improved natural language processing. Expert Systems with Applications, v. 181, 115113, 2021. DOI: https://doi.org/10.1016/j.eswa.2021.115113

SU, Haibo; ZHANG, Yi; DING, Ying. Exploring technological trends through text mining and natural language processing. Journal of Informetrics, v. 13, n. 3, p. 763–777, 2019. DOI: https://doi.org/10.1016/j.joi.2019.07.001

SYED, Rizwan; KHAN, Atif; RAZA, Asim. Robotic process automation: contemporary themes and challenges. Computers in Industry, v. 115, 103162, 2020. DOI: https://doi.org/10.1016/j.compind.2019.103162

ZHOU, Xiaofeng; LI, Ming; WANG, Hui. Efficient data storage and exchange with JSON in big data pipelines. Future Generation Computer Systems, v. 108, p. 658–667, 2020. DOI: https://doi.org/10.1016/j.future.2020.03.005

VARA, Pedro Augusto. MBA_IA_BIG_DATA_USP_2025 [repositório]. GitHub, 2025. Disponível em: https://github.com/pedrovara/MBA_IA_BIG_DATA_USP_2025

. Acesso em: 29 out. 2025.

GLOSSÁRIO

Aprendizado de Máquina (*Machine Learning*) – Subárea da Inteligência Artificial que desenvolve algoritmos capazes de aprender padrões e realizar previsões a partir de dados.

BDTA (Biblioteca Digital de Trabalhos Acadêmicos) – Repositório institucional da Universidade de São Paulo que reúne teses, dissertações e monografias.

Big Data – Conjunto de métodos e tecnologias voltados à coleta, armazenamento e análise de grandes volumes de dados estruturados e não estruturados.

Bigrama (*Bigram*) – Sequência de duas palavras consecutivas em um texto, usada para modelar coocorrências linguísticas.

Dice Coefficient – Métrica estatística de similaridade entre cadeias de caracteres, utilizada para correspondência aproximada de termos.

JSON (*JavaScript Object Notation*) – Formato leve de intercâmbio de dados estruturados, usado para armazenar resultados e metadados do *pipeline*.

Lematização (*Lemmatization*) – Processo de reduzir palavras à sua forma base (lema), preservando o sentido semântico.

N-grama (*N-gram*) – Sequência de n palavras consecutivas, empregada para capturar relações e padrões locais em textos.

NLTK (*Natural Language Toolkit*) – Biblioteca *Python* para processamento de linguagem natural, utilizada nas etapas de tokenização, *stemming* e filtragem.

Pipeline – Fluxo sequencial de processamento automatizado composto por múltiplas etapas interdependentes.

PLN (Processamento de Linguagem Natural) – Área da Inteligência Artificial que busca permitir que máquinas compreendam e processem a linguagem humana.

Stemming – Técnica de redução de palavras a seus radicais morfológicos, eliminando sufixos e flexões.

TF-IDF (*Term Frequency – Inverse Document Frequency*) – Métrica que mede a importância de um termo em um conjunto de documentos.

Transformer – Arquitetura de rede neural baseada em mecanismos de atenção, amplamente utilizada em modelos de linguagem natural.

UiPath – Plataforma de automação robótica de processos (*RPA*) utilizada para coleta automática dos arquivos da BDTA.

WordCloud – Visualização gráfica que representa a frequência de termos em um corpus textual, com o tamanho da palavra proporcional à sua ocorrência.

Wordlist – Lista controlada de termos e expressões utilizadas para filtrar e identificar tecnologias nos textos analisados.

Apêndice A – Script prolog.py (Montagem e Conexão)

```
from google.colab import drive
drive.mount('/content/drive')

PATH_PREFIX = "./drive/MyDrive/MBA_PRATICO_USP_IA/"

PDF_DIR = PATH_PREFIX + "/PDFs"
OUTPUT_DIR = PATH_PREFIX + "/extracted"
WORDLIST_PATH = PATH_PREFIX + "/wordlist.txt"

DICE_THRESHOLD = 0.80

MULTITHREADING = False
MULTITHREADING_CPUS = 6

FULL_WORDCLOUD_FILE = PATH_PREFIX + "/full-wordcloud.png"
PART_WORDCLOUD_PREFIX = PATH_PREFIX + "/year-wordcloud."

import os
import string
import json
import re
import time
import math

!pip install PyMuPDF
```

```
import nltk


nltk.download('stopwords')
nltk.download('punkt')
nltk.download('punkt_tab')
```

Apêndice B – Script phase_1.py (Processamento Principal)

```
        def load_wordlist(path):
"""Load wordlist with both original and stemmed forms."""
with open(path, "r", encoding="utf-8") as f:
    lines = [line.strip().lower() for line in f if line.strip()]


single_words_orig = set()
single_words_stem = set()
phrases_orig = set()
phrases_stem = set()


for line in lines:
    tokens = nltk.word_tokenize(line)
    if len(tokens) == 1:
        word = tokens[0]
        single_words_orig.add(word)
        single_words_stem.add(stemmer.stem(word))
    else:
        phrase = " ".join(tokens)
        stemmed_phrase = " ".join(stemmer.stem(w) for w in tokens)
        phrases_orig.add(phrase)
        phrases_stem.add(stemmed_phrase)
```

```
    return {
        "single_orig": single_words_orig,
        "single_stem": single_words_stem,
        "phrases_orig": phrases_orig,
        "phrases_stem": phrases_stem
    }

def filter_words_by_wordlist(words, wordlist_dict):
    """Match original or stemmed words from document against wordlist."""
    matched = []
    for word in words:
        stem = stemmer.stem(word)
        if word in wordlist_dict["single_orig"] or stem in wordlist_dict["single_stem"]:
            matched.append(word)
    return matched

def generate_ngrams(word_list, n=2):
    """Generate n-grams (bigrams/trigrams) from word list."""
    return [" ".join(gram) for gram in ngrams(word_list, n)]

def match_ngrams_to_wordlist(ngrams_list, wordlist_dict):
    """Match n-grams to original and stemmed wordlist phrases."""
```

```
    matched = []
    for ngram in ngrams_list:
        if ngram in wordlist_dict["phrases_orig"] or ngram in wordlist_dict["phrases_stem"]:
            matched.append(ngram)
    return matched


def dice_coefficient(a, b):
    """Calculate Dice coefficient between two strings."""
    def bigrams(s):
        return set(ngrams(s, 2))
    a_bigrams = bigrams(a)
    b_bigrams = bigrams(b)
    if not a_bigrams or not b_bigrams:
        return 0.0
    overlap = len(a_bigrams & b_bigrams)
    return 2 * overlap / (len(a_bigrams) + len(b_bigrams))


def match_by_dice(words, wordlist_dict, threshold=DICE_THRESHOLD):
    """Fuzzy match words using Dice coefficient ≥ threshold."""
    all_targets = wordlist_dict["single_orig"] | wordlist_dict["single_stem"]
    matches = set()
    for word in words:
        for target in all_targets:
```

```
            if dice_coefficient(word, target) >= threshold:
                matches.add(word)
                break
    return list(matches)


def extract_publication_year(pdf_path, max_pages=10):
    """Extract the first 4-digit year found in the first N pages of the PDF."""
    year_pattern = re.compile(r"\b(19|20)\d{2}\b")
    with fitz.open(pdf_path) as doc:
        pages_to_scan = min(max_pages, doc.page_count)
        for i in range(pages_to_scan):
            text = doc[i].get_text()
            match = year_pattern.search(text)
            if match:
                return match.group(0)
    return None


def extract_text_from_pdf(pdf_path):
    """Extract text from a PDF using PyMuPDF."""
    text = ""
    with fitz.open(pdf_path) as doc:
        for page in doc:
            text += page.get_text()
```

```
    return text


def extract_metadata(pdf_path):
    """Extract metadata and resource_date (first year found on initial pages)."""
    with fitz.open(pdf_path) as doc:
        metadata = doc.metadata
    resource_date = extract_publication_year(pdf_path)
    return {
        "title": metadata.get("title"),
        "author": metadata.get("author"),
        "creation_date": metadata.get("creationDate"),
        "resource_date": resource_date
    }


def clean_text(text):
    text = text.lower().translate(str.maketrans("", "", string.punctuation))
    words = nltk.word_tokenize(text)
    return words


def stem_text(words):
    filtered = [stemmer.stem(word) for word in words if word not in stop_words and word not
in stop_words_pt and word.isalpha()]
    return filtered
```

```
def process_all_pdfs (startat=0, endat=1000, pdf_dir=PDF_DIR, cpu_id=-1):
    #startat = parm[0]
    #endat = parm[1]
    #pdf_dir = parm[2]
    #cpu_id = parm[3]
    counter = 0
    for filename in os.listdir(pdf_dir):

        counter += 1
        if counter >= endat:
            break
        if counter < startat:
            continue

        if filename.endswith(".pdf"):
            t_origin = time.time()
            pdf_path = os.path.join(pdf_dir, filename)
            print(f"\n[{cpu_id}] [+] {filename}")

            # Extract text and metadata
            raw_text = extract_text_from_pdf(pdf_path)
```

```
        t0 = time.time()
        metadata = extract_metadata(pdf_path)
        print(f" [{cpu_id}] + Resource_Date: {metadata['resource_date']} ({time.time()-t0:.3f}s)")

        t0 = time.time()
        all_words = clean_text(raw_text)
        stemmed_words = stem_text(all_words)
        # deduplicate and remove false-words (like urls: httpsomesitecom)
        stemmed_words = list(set(sw.lower() for sw in stemmed_words if len(sw)<=16 and len(sw)>=5))
        print(f" [{cpu_id}] + Word_Count: {len(stemmed_words)} ({time.time()-t0:.3f}s)")

        # Create output object
        result = {
            "file": filename,
            "metadata": metadata,
            "word_count": len(stemmed_words)
            # "stemmed_words": stemmed_words
        }

        wordlist_dict = load_wordlist(WORDLIST_PATH)

        # Step 1 & 2: Wordlist match (original + stemmed)
```

```
        t0 = time.time()
        wordlist_matches = filter_words_by_wordlist(stemmed_words, wordlist_dict)
        print(f" [{cpu_id}] + Wl_Match_Count: {len(wordlist_matches)} ({time.time()-t0:.3f}s)")

        # Step 3: Generate bigrams and trigrams
        t0 = time.time()
        bigrams_list = generate_ngrams(stemmed_words, 2)
        print(f" [{cpu_id}] + Bigram_List_Size: {len(bigrams_list)} ({time.time()-t0:.3f}s)")

        t0 = time.time()
        trigrams_list = generate_ngrams(stemmed_words, 3)
        print(f" [{cpu_id}] + Trigram_List_Size: {len(trigrams_list)} ({time.time()-t0:.3f}s)")

        # Step 4: Match n-grams to wordlist
        t0 = time.time()
        matched_bigrams = match_ngrams_to_wordlist(bigrams_list, wordlist_dict)
        print(f" [{cpu_id}] + Bigram_Match_Count: {len(matched_bigrams)} ({time.time()-t0:.3f}s)")

        t0 = time.time()
        matched_trigrams = match_ngrams_to_wordlist(trigrams_list, wordlist_dict)
```

```
        print(f" [{cpu_id}] + Trigram_Match_Count: {len(trigrams_list)} ({time.time()-t0:.3f}s)")

        # Step 5: Dice similarity matching
        t0 = time.time()
        dice_matches = match_by_dice(stemmed_words, wordlist_dict)
        print(f" [{cpu_id}] + Dice_Match_Count: {len(dice_matches)} ({time.time()-t0:.3f}s)")

        # Append to JSON structure
        result["step_b_data"] = {
            "wordlist_matches": wordlist_matches,
            "count": len(wordlist_matches)
        }

        result["step_c_data"] = {
            "ngrams": {
                "bigrams": matched_bigrams,
                "trigrams": matched_trigrams
            },
            "dice_matches": dice_matches,
            "counts": {
                "matched_bigrams": len(matched_bigrams),
                "matched_trigrams": len(matched_trigrams),
```

```
            "dice_matches": len(dice_matches)
        }
    }


    # Step 6: Stemming restoration
    t0 = time.time()

    candidates = set(wordlist_matches) | set(dice_matches)
    restored_words = {
        (c, wword)
        for c in candidates
        for wword in all_words
        if wword.startswith(c)
    }

    print(f" [{cpu_id}] + Stemming_Restore: {len(restored_words)} ({time.time()-t0:.3f}s)")

    result["restored"] = {
        "count": len(restored_words),
        "words": list(restored_words)
    }
```

```
        # Save to ./extracted/
        json_filename = os.path.splitext(filename)[0] + ".json"
        with open(os.path.join(OUTPUT_DIR, json_filename), "w", encoding="utf-8") as f:
            json.dump(result, f, indent=2, ensure_ascii=False)

        print(f" [{cpu_id}] +  Elapsed_Time:   {time.time()-t_origin:.3f}s")

from collections import Counter
from wordcloud import WordCloud
import matplotlib.pyplot as plt
import os
import fitz  # PyMuPDF
import nltk
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.util import ngrams
import string
import json
import re
import time
import math
from multiprocessing import Pool, Process
```

```
# Setup was already preloaded on first run
# nltk.download('stopwords')
# nltk.download('punkt_tab')

stop_words = set(stopwords.words('english'))
stop_words_pt = set(stopwords.words('portuguese'))
stemmer = PorterStemmer()

#PDF_DIR = "./PDFs"
#OUTPUT_DIR = "./resources/extracted/"
#PDF_DIR = "/media/void/10bc5bcd-2968-4970-8e1b-e0c62c496772/pedro/pdfs/CHUNK"
#OUTPUT_DIR = "/media/void/10bc5bcd-2968-4970-8e1b-e0c62c496772/pedro/extracted"

#WORDLIST_PATH = "./resources/wordlist.txt"
os.makedirs(OUTPUT_DIR, exist_ok=True)

if MULTITHREADING:
    MULTITHREADING_UNITS_PER_PROCESSOR = math.ceil(len(os.listdir(PDF_DIR)) / MULTITHREADING_CPUS)
    pool = Pool(processes=MULTITHREADING_CPUS)
    iterable_args = [(x, x + MULTITHREADING_UNITS_PER_PROCESSOR, PDF_DIR, i) for i, x in enumerate(range(0, len(os.listdir(PDF_DIR)), MULTITHREADING_UNITS_PER_PROCESSOR))]
```

```
    for i in range(MULTITHREADING_CPUS):
        p = Process(target=process_all_pdfs, args = iterable_args[i])
        p.start()
else:
    process_all_pdfs(0, len(os.listdir(PDF_DIR)), PDF_DIR)
```

Apêndice C – Script phase_2.py (Geradores de Saídas)

```
from collections import Counter

from wordcloud import WordCloud

import matplotlib.pyplot as plt

import os

import string

import json

import re

import time

import math

EXTRACTED_DIR = OUTPUT_DIR

OUTPUT_FILE = FULL_WORDCLOUD_FILE


def collect_words():

    """Read all JSON files and collect words from step_b_data.wordlist_matches."""

    all_words = []

    all_words_stem = []


    for filename in os.listdir(EXTRACTED_DIR):

        if filename.endswith(".json"):

            filepath = os.path.join(EXTRACTED_DIR, filename)

            with open(filepath, "r", encoding="utf-8") as f:
```

```
            data = json.load(f)

        matches = data.get("restored", {}).get("words", [])
        for (word_a, word_b) in matches:
            # Normalize: lowercase, replace spaces with "-"
            norm_a = word_a.lower().replace(" ", "-")
            norm_b = word_b.lower().replace(" ", "-")
            all_words.append(norm_a)
            all_words_stem.append(norm_b)

    return (all_words, all_words_stem)

def generate_wordcloud(words, top_n=50, output_path=OUTPUT_FILE):
    """Generate and save a word cloud with top N words."""
    #counter = Counter(words[0])
    #most_common = dict(counter.most_common(top_n))
    #most_common = counter.most_common(top_n)

    # Step 1: count most common fragments
    top_fragments = Counter(words[0]).most_common(30)

    # Step 2: for each fragment, get indexes in listA
    all_words = {}
```

```
result = {}
for fragment, count in top_fragments:
    indexes = [i for i, val in enumerate(words[0]) if val == fragment]
    #full_words = list(set([words[0][i] for i in indexes])) # THIS ONE GENERATES A STEMMED GRAPH (nice)
    full_words = list(set([words[1][i] for i in indexes])) # THIS ONE GENERTES A REGULAR GRAPH (less nice)
    result[fragment] = {
        "count": count,
        #"indexes": indexes,
        #"full_words": full_words[0]
    }
    #for word in full_words:
    all_words[full_words[0]] = count

wc = WordCloud(
    width=1200,
    height=800,
    background_color="white"
).generate_from_frequencies(all_words)

plt.figure(figsize=(12, 8))
plt.imshow(wc, interpolation="bilinear")
plt.axis("off")
```

```
    plt.tight_layout()
    plt.savefig(output_path, format="png", dpi=150)
    plt.close()

    print(result)
    print(f"Word cloud saved to {output_path}")

if __name__ == "__main__":
    words_pair = collect_words()
    print(len(words_pair[0]))
    print(words_pair)
    if not words_pair:
        print("No words found in step_b_data.wordlist_matches.")
    else:
        generate_wordcloud(words_pair, top_n=50)

OUTPUT_EXTENSION = ".png"
EXTRACTED_DIR = OUTPUT_DIR
OUTPUT_FILE = PART_WORDCLOUD_PREFIX

batch_years = [
    [1970, 1974],
    [1975, 1979],
```

```
    [1980, 1984],
    [1985, 1989],
    [1990, 1994],
    [1995, 1999],
    [2000, 2004],
    [2005, 2009],
    [2010, 2014],
    [2015, 2019],
    [2020, 2021],
    [2021, 2022],
    [2023, 2024],
    [2024, 2025],
]

def collect_words(year_start, year_end):
    """Read all JSON files and collect words from step_b_data.wordlist_matches."""
    all_words = []
    all_words_stem = []

    for filename in os.listdir(EXTRACTED_DIR):
        if filename.endswith(".json"):
            filepath = os.path.join(EXTRACTED_DIR, filename)
            with open(filepath, "r", encoding="utf-8") as f:
```

```
        data = json.load(f)


data_s1 = data.get("metadata", {})
data_s2 = data_s1.get("resource_date", {})
if not data_s2:
    print(f" - Warn, no data_s2 on {filename}, assuming range 2024-2025")
    data_s2 = 2025


unit_year = int(data_s2)
if unit_year < year_start or unit_year > year_end:
    continue


matches = data.get("step_b_data", {}).get("wordlist_matches", [])
for word_a in matches:
    # Normalize: lowercase, replace spaces with "-"
    norm_a = word_a.lower().replace(" ", "-")
    all_words.append(norm_a)
    all_words_stem.append(norm_a)
# matches = data.get("restored", {}).get("words", [])
# for (word_a, word_b) in matches:
#     # Normalize: lowercase, replace spaces with "-"
#     norm_a = word_a.lower().replace(" ", "-")
#     norm_b = word_b.lower().replace(" ", "-")
```

```
        #     all_words.append(norm_a)
        #     all_words_stem.append(norm_b)

    if len(all_words) == 0:
        return None

    return (all_words, all_words_stem)

def generate_wordcloud(words, top_n=50, output_path=OUTPUT_FILE):
    """Generate and save a word cloud with top N words."""
    #counter = Counter(words[0])
    #most_common = dict(counter.most_common(top_n))
    #most_common = counter.most_common(top_n)

    # Step 1: count most common fragments
    top_fragments = Counter(words[0]).most_common(20)

    # Step 2: for each fragment, get indexes in listA
    all_words = {}
    result = {}
    print(top_fragments)
    for fragment, count in top_fragments:
        indexes = [i for i, val in enumerate(words[0]) if val == fragment]
```

```
    full_words = list(set([words[1][i] for i in indexes]))
    result[fragment] = {
        "count": count,
        #"indexes": indexes,
        #"full_words": full_words[0]
    }
    #for wword in full_words:
    all_words[full_words[0]] = count

wc = WordCloud(
    width=1200,
    height=800,
    background_color="white"
).generate_from_frequencies((all_words))

plt.figure(figsize=(12, 8))
plt.imshow(wc, interpolation="bilinear")
plt.axis("off")
plt.tight_layout()
plt.savefig(output_path, format="png", dpi=150)
plt.close()

print(f"Word cloud saved to {output_path}")
```

```
if __name__ == "__main__":
    for pair in batch_years:
        words_pair = collect_words(pair[0], pair[1])
        if not words_pair:
            print("No words found in step_b_data.wordlist_matches for batch years [" + str(pair[0]) + '-' + str(pair[1]) + "]")
            words_pair = collect_words(2024, 2025)

        generate_wordcloud(words_pair, top_n=50, output_path=OUTPUT_FILE + str(pair[0]) + '-' + str(pair[1]) + OUTPUT_EXTENSION)

import os
import json
from collections import Counter, defaultdict
import matplotlib.pyplot as plt
from nltk.util import ngrams

EXTRACTED_DIR = OUTPUT_DIR + "2"
OUTPUT_FILE = PATH_PREFIX + "top_words_per_year3.png"
MIN_YEAR = 2000  # ignore anything before this year
YEAR_INTERVAL = 1  # group years by this interval
WORDLIST2 = PATH_PREFIX + "/wl2.txt"
with open(WORDLIST2, "r", encoding="utf-8") as f:
```

```
    wordlist2 = f.read().splitlines()


def dice_coefficient(a, b):
    """Calculate Dice coefficient between two strings."""
    def bigrams(s):
        return set(ngrams(s, 2))
    a_bigrams = bigrams(a)
    b_bigrams = bigrams(b)
    if not a_bigrams or not b_bigrams:
        return 0.0
    overlap = len(a_bigrams & b_bigrams)
    return 2 * overlap / (len(a_bigrams) + len(b_bigrams))


def round_to_5year_group(year):
    """Round a given year down to the nearest multiple of YEAR_INTERVAL."""
    return int(math.floor(year / YEAR_INTERVAL) * YEAR_INTERVAL)


def collect_words_by_year():
    """Read all JSON files and collect normalized words grouped by 5-year periods."""
    words_by_year = defaultdict(list)

    for filename in os.listdir(EXTRACTED_DIR):
        if not filename.endswith(".json"):
```

```
    continue

filepath = os.path.join(EXTRACTED_DIR, filename)
with open(filepath, "r", encoding="utf-8") as f:
    data = json.load(f)

year_value = data.get("metadata", {}).get("resource_date", "unknown")

# Try to parse the year
try:
    year = int(str(year_value))
    if year < MIN_YEAR:
        continue  # skip years before 1980
    year_group = str(round_to_5year_group(year))
except (ValueError, TypeError):
    year_group = "unknown"

matches = data.get("step_b_data", {}).get("wordlist_matches", [])
#matches = data.get("step_b_data", {}).get("dice_matches", [])

matches2 = []

for w in matches:
```

```
            for w2 in wordlist2:
              tmpres = dice_coefficient(w, w2)
              if tmpres >= 0.8:
                matches2.append(w)

          normalized = [w.lower().replace(" ", "-") for w in matches2]
          words_by_year[year_group].extend(normalized)

      return words_by_year

def get_top_words_per_year(words_by_year, top_n=3):
    """Get top N words per 5-year group using Counter."""
    top_per_year = {}
    for year, words in words_by_year.items():
        counter = Counter(words)
        top_per_year[year] = counter.most_common(top_n)
    return top_per_year

def plot_top_words_per_year(top_per_year, output_path=OUTPUT_FILE):
    """Generate and save a bar chart of top words per 5-year period."""
    plt.figure(figsize=(50, 15))
    plt.title("Top Most Common Words per Year")
    plt.xlabel("Year")
```

```
plt.ylabel("Frequency")


# Sort 5-year groups numerically, keep 'unknown' last
numeric_years = sorted([y for y in top_per_year if y.isdigit()], key=int)
if "unknown" in top_per_year:
    sorted_years = numeric_years + ["unknown"]
else:
    sorted_years = numeric_years


#bar_width = 0.25
bar_width = 0.05
#colors = ["tab:blue", "tab:orange", "tab:green", "tab:blue", "tab:orange", "tab:green", "tab:blue", "tab:orange", "tab:green", "tab:blue", "tab:orange", "tab:green"]
#colors = ["tab:blue", "tab:orange", "tab:green"]
colors = [
"tab:blue",
"tab:orange",
"tab:green",
"tab:red",
"tab:purple",
"tab:brown",
"tab:pink",
"tab:gray",
```

```
        "tab:olive",
        "tab:cyan"
    ]
    #colors = colors * 20

    for i, (word_rank, color) in enumerate(zip(range(10), colors)):
        words = []
        freqs = []
        for year in sorted_years:
            year_data = top_per_year[year]
            if len(year_data) > word_rank:
                word, freq = year_data[word_rank]
            else:
                word, freq = "", 0
            words.append(word)
            freqs.append(freq)

        x_positions = [x + i * bar_width for x in range(len(sorted_years))]
        plt.bar(x_positions, freqs, width=bar_width, color=color, label=f"Top {i+1}")

        # Add labels
        for x, f, w in zip(x_positions, freqs, words):
            if f > 0:
```

```
            plt.text(x, f + 0.5, w, ha="center", rotation=90, fontsize=9)


    plt.xticks(
        [r + bar_width for r in range(len(sorted_years))],
        sorted_years
    )
    plt.legend()
    #plt.tight_layout()
    plt.savefig(output_path, dpi=300)
    plt.close()
    print(f"Bar chart saved to {output_path}")


if __name__ == "__main__":
    words_by_year = collect_words_by_year()
    if not words_by_year:
        print("No valid data found in step_b_data.wordlist_matches.")
    else:
        top_per_year = get_top_words_per_year(words_by_year, top_n=43)
        plot_top_words_per_year(top_per_year)
```

Apêndice D – Gráfico de Barras Originalmente Gerado pelo Pipeline