Universidade de Brasília

Programa de Pós-Graduação em Direito, Políticas Públicas e Regulação

Eloi Ricardo Reffatti

Ferramenta para Extração de Dados do Portal do Supremo Tribunal: Relatório Técnico de Desenvolvimento

Relatório técnico apresentado como requisito para obtenção do título de Mestre em Direito do Programa de Pós-Graduação Profissional em Direito da Universidade de Brasília, área de concentração “Direito, Regulação e Políticas públicas”.

Linha de Pesquisa: Pesquisa e Desenvolvimento.

Orientador: Prof. Henrique Araújo Costa

Brasília – DF

2026

ELOI RICARDO REFFATTI

Desenvolvimento de Ferramenta para Extração de Dados do Portal do Supremo Tribunal

Relatório técnico apresentado como requisito para obtenção do título de Mestre em Direito do Programa de Pós-Graduação Profissional em Direito da Universidade de Brasília, área de concentração “Direito, Regulação e Políticas públicas”.

Em 24 de abril de 2026, o candidato foi considerado aprovado pela Banca Examinadora.

Banca Examinadora:

_____

Prof. Dr. Henrique Araújo Costa

(Orientador – Presidente)

_____

Profª. Drª. Taynara Tiemi Ono

(Membro Externo)

_____

Prof. Dr. Amilar Domingos Moreira Martins

(Membro externo)

_____

Prof. Dr. Wilson Roberto Theodoro Filho

(Suplente)

RESUMO

Este relatório técnico apresenta o desenvolvimento de uma aplicação modular de código aberto em Python destinada à raspagem automática de dados do portal do Supremo Tribunal Federal, com utilização da biblioteca *Selenium*. Os dados obtidos são enriquecidos por meio da *API DataJud*, disponibilizada pelo Conselho Nacional de Justiça, e submetidos a análise baseada em algoritmos heurísticos. Posteriormente, as informações são consolidadas e disponibilizadas nos formatos JSON e XLSX. A aplicação incorpora, ainda, uma interface gráfica desenvolvida com a biblioteca *Tkinter*, concebida para garantir a acessibilidade da ferramenta a pesquisadores sem experiência prévia em programação.

Palavras-chave: raspagem de dados jurídicos; Supremo Tribunal Federal; automação; aplicação em Python; *Selenium*; enriquecimento de dados; código aberto; metadados judiciais; jurimetria.

ABSTRACT

This technical report presents the development of a modular open-source Python application designed for the automatic scraping of data from the Supreme Federal Court's portal, using the Selenium library. The data obtained are enriched through the DataJud API, provided by the National Council of Justice, and subjected to analysis based on heuristic algorithms. Subsequently, the information is consolidated and made available in JSON and XLSX formats. The application also incorporates a graphical interface developed with the Tkinter library, designed to ensure accessibility of the tool to researchers without prior programming experience.

Keywords: legal data scraping; Supreme Federal Court; automation; Python application; Selenium; data enrichment; open source; judicial metadata; jurimetrics.

DEDICATÓRIA

Dedico este trabalho à minha filha, Isabela Joaquina, que, apesar da tenra idade, enfrentou longos períodos de afastamento do convívio paterno, sem que pudesse compreender plenamente as razões dessa ausência.

Rendo igualmente minha homenagem a Veridiana da Mata, pelo apoio e incentivo oferecidos durante o processo de elaboração do código e do relatório técnico. Sua colaboração permaneceu constante e essencial, sobretudo no cuidado com nossa filha, contribuindo significativamente para a atenuação dos impactos decorrentes de minha ausência.

AGRADECIMENTOS

Agradeço a todos os colegas discentes do programa de pós-graduação, que proporcionaram debates de altíssimo nível durante as aulas e os seminários apresentados nas disciplinas que cursamos em conjunto, contribuindo para que a pós-graduação refletisse o mesmo espírito de corpo que, felizmente, vivencio há mais de dezesseis anos no âmbito do Supremo Tribunal Federal, onde o trabalho em equipe, a colaboração e o respeito mútuo são evidentes.

Expresso, igualmente, minha gratidão aos professores do programa, cujas disciplinas tive o privilégio de cursar, por compartilharem com maestria o conhecimento acumulado em suas respectivas trajetórias acadêmicas, por intermediarem debates sobre temas de elevado interesse científico e por guiarem o desenvolvimento de pesquisas empíricas sólidas: Dra. Maria Pia dos Santos Lima Guerra Dalledone, Dr. Marcio Nunes Iorio Aranha Oliveira, Dr. Othon de Azevedo Lopes e Dr. Fabiano Hartmann Peixoto.

Dirijo um agradecimento especial ao meu orientador, Dr. Henrique Araújo Costa, pela dedicação, pela compreensão do sentido deste trabalho, pelo incentivo ao seu desenvolvimento e pelas valiosas sugestões oferecidas durante a execução desta pesquisa.

Ao Dr. Alexandre Araújo Costa registro meus agradecimentos pela ênfase com que destacou a importância da modelagem de dados no decorrer das disciplinas de Metodologia Científica e de Análise de Dados, apresentando, ainda, aos seus alunos a linguagem Python como uma relevante ferramenta de pesquisa e análise.

Por fim, agradeço ao Supremo Tribunal Federal por incentivar a qualificação profissional do seu quadro de servidores.

Sumário

1 INTRODUÇÃO

Este relatório técnico descreve um sistema completo de extração, enriquecimento, processamento e análise de dados relativos a processos judiciais em tramitação ou já encerrados no Supremo Tribunal Federal (STF). A solução foi implementada em Python e organizada em cinco componentes integrados: quatro módulos de processamento (extração, enriquecimento, consolidação e biblioteca de suporte) e uma interface gráfica (GUI) destinada à parametrização e execução assistida pelo usuário. Esses componentes operam de maneira coordenada para cumprir três tarefas distintas e complementares: (i) raspagem de dados no portal de 9consulta processual do STF (*web scraping*); (ii) enriquecimento das informações extraídas mediante consulta à API pública do Conselho Nacional de Justiça (*DataJud*), instituída pela Resolução CNJ nº 331/2020; e (iii) consolidação dos dados em formato estruturado para análise.

O primeiro componente é o ModuloSTFSelenium.py, que centraliza estruturas de dados e funções reutilizadas pelos demais scripts, atuando como uma biblioteca de suporte e reduzindo a repetição de código. O módulo STF_Selenium_v3.0.py executa a raspagem de dados no portal do STF por meio da automação do navegador Google Chrome via *Selenium/WebDriver*, acessando as páginas de consulta processual e extraindo informações em formato estruturado. A extração foi concebida para operar com diferentes classes processuais, minimizando a necessidade de adaptações específicas para cada tipo de processo de competência do STF.

O módulo Modulo_API_DataJud_v3.py realiza requisições à API pública do *DataJud* para complementar os dados coletados no portal do STF. Sua finalidade específica é obter informações sobre a tramitação e a classe processual na instância de origem, além do assunto do processo cadastrado nos bancos do CNJ, quando disponível.

O módulo Mon_Plan_c_dados_API.py, por sua vez, consolida os dados a partir da leitura dos arquivos gerados pelos módulos anteriores e executa análises automatizadas, tais como: identificação e separação de sujeitos processuais (partes

e procuradores) por polo processual (ativo, passivo e terceiros[1]); contagem de andamentos, decisões e recursos internos; detecção de julgamentos virtuais e de pedidos de vista/destaque; indicação da primeira decisão proferida no processo; e verificação de eventual reforma provocada por recursos internos, incluindo a quantificação dessas reformas por processo.

A organização em módulos visa facilitar a manutenção e a evolução do sistema. Apesar de robusto, o software é inerentemente sensível a dependências técnicas externas: (i) depende de bibliotecas de terceiros — de modo que atualizações podem exigir ajustes no código, sob pena de inoperância — e (ii) é sensível a alterações no portal do STF, decorrentes de atualizações implementadas pela Secretaria de Tecnologia da Informação do Tribunal, o que pode demandar revisão periódica dos seletores (por exemplo, XPaths) utilizados na extração.

[1] A terminologia *"terceiros"* não é utilizada aqui no sentido estritamente técnico em relação à intervenção de terceiros, pois não assumem posição ordinária de parte no processo, como no caso da assistência, simples ou litisconsorcial, do chamamento ao processo ou do *"amici curiae"*. A aplicação do termo assume um sentido excludente, de modo que quem não integra o polo ativo ou o passivo é considerado terceiro, em função do cadastramento feito pelo próprio STF em caso como da AP 2.698/DF, na qual a Polícia Federal foi cadastrada como interessada, embora sem assumir propriamente o atributo de parte na relação processual.

2. HISTÓRICO DE DESENVOLVIMENTO

2.1 Motivação

Às vésperas do processo seletivo relacionado ao contexto do programa, a Corte Suprema publicou o Edital de Chamamento Público nº 001/2023[2], cujo objetivo era estimular o desenvolvimento de protótipos de ferramentas de inteligência artificial para criação de sumários automatizados de processos judiciais no âmbito do Tribunal. Em outras palavras, pretendia-se obter um protótipo de sistema de IA generativa capaz de produzir resumos para as classes Recurso Extraordinário (RE) e Recurso Extraordinário (ARE).

Embora a Universidade de Brasília não tenha inscrito proposta para participar do chamamento — em razão do envolvimento da coordenação do Programa de Mestrado Profissional em Direito na etapa de seleção de candidatos —, o STF posteriormente disponibilizou o conjunto de dados fornecido aos participantes do certame. Nesse contexto, no âmbito do Programa de Mestrado Profissional em Direito da Universidade de Brasília, formou-se um grupo de trabalho envolvendo parcela significativa dos mestrandos da turma de 2023 do STF.

Em síntese, o grupo buscava testar grandes modelos de linguagem (*LLMs — Large Language Models*) e desenvolver um *prompt* suficientemente adequado ao desenvolvimento de um sistema capaz de redigir o relatório do processo, em padrão compatível com a primeira parte de uma decisão judicial, de modo a poupar tempo da assessoria e da equipe técnica e permitir maior dedicação à pesquisa da solução da controvérsia jurídica subjacente. Para tanto, o grupo analisou processos reais cujos autos foram cedidos pelo STF.

Em outubro de 2024, surgiram modelos de linguagem com resultados significativamente superiores aos de versões anteriores, o que permitiu avanços também na etapa de exame do recurso. A partir de uma pequena alteração no prompt em desenvolvimento, realizou-se teste no qual se solicitou que o sistema avançasse além do relatório e emitisse análise para além dessa seção. Em seguida, a decisão

[2] *Vide* URL < https://noticias.stf.jus.br/postsnoticias/stf-faz-chamamento-publico-para-projetos-de-inteligencia-artificial-que-automatizem-resumos-de-processos/ >, acesso em 30/10/2025.

efetivamente proferida no processo foi consultada no sítio eletrônico do STF. **Surpreendentemente**, naquele caso, os fundamentos gerados pelo sistema coincidiram em grande parte com aqueles da decisão real, segundo a qual a controvérsia não se revestia de caráter constitucional.

Todavia, em relação àquele recurso específico, surgiu debate: um integrante do grupo sustentou que não se deveria ter avançado ao exame da constitucionalidade, diante de possível defeito formal do recurso. Segundo esse entendimento, o recorrente não teria demonstrado suficientemente a repercussão geral da controvérsia jurídica; vale dizer, à luz da jurisprudência do STF, a preliminar de repercussão geral não estaria suficientemente fundamentada. A divergência de interpretações levou à formulação da seguinte pergunta de pesquisa: o STF (ou seus Ministros, considerando que a maioria das decisões desse tipo é monocrática) adota critérios mais ou menos objetivos para não conhecer recurso extraordinário por deficiência ou insuficiência de fundamentação da preliminar de repercussão geral?

Para responder ao questionamento, delineou-se metodologia que, em primeiro lugar, demandava a identificação de recursos extraordinários e recursos extraordinários com agravo não conhecidos com fundamento na deficiência ou na insuficiência da preliminar de repercussão geral. Em segundo lugar, seria necessário identificar recursos cujo mérito tivesse sido analisado ou cujo seguimento tivesse sido negado por óbices não formais – por entender-se, por exemplo, que a controvérsia é infraconstitucional, exige reexame de fatos e provas ou envolve interpretação de cláusulas contratuais. Na sequência, identificar-se-iam recursos com repercussão geral reconhecida e aqueles em que a Corte recusou a repercussão geral da controvérsia suscitada. Por fim, proceder-se-ia à análise das preliminares de repercussão geral apresentadas pelos recorrentes, buscando padrões indicativos de critérios mais ou menos objetivos que justificassem: (1) a recusa de processamento por deficiência/insuficiência da preliminar; (2) a superação desse óbice e o avanço no exame da controvérsia; e (3) a formação de juízo positivo ou negativo quanto à repercussão geral da controvérsia jurídica objeto do recurso.

Após leitura do trabalho “A Quem Interessa o Controle Concentrado de Constitucionalidade? — O Descompasso entre Teoria e Prática na Defesa dos Direitos

Fundamentais"[3], de Alexandre Araújo Costa e Juliano Benvindo, e das aulas de Ciência de Dados, ministradas pelo primeiro, o autor deste trabalho tomou conhecimento também de iniciativas de desenvolvimento de sistemas automatizados de extração de dados processuais em Python, com repositórios públicos na plataforma GitHub, inclusive com códigos escritos pelo Dr. Alexandre Araújo Costa [4]. Com inspiração nesses códigos, decidiu-se desenvolver sistema próprio, em Python, voltado especialmente às classes ARE e RE.

O sistema automatizado deveria consultar as páginas de andamentos processuais de um intervalo ou de uma lista pré-definidos recursos e raspar todos os dados disponíveis (número processual, partes, advogados, datas dos andamentos *et cetera*). Considerando que o STF disponibiliza, na página de andamentos, *links* para decisões proferidas nos processos, planejou-se que o sistema abrisse cada decisão publicada e verificasse o resultado do julgamento e seus fundamentos. Isso exigiria capacidade de abrir documentos, ler arquivos PDF e identificar dispositivo e fundamentos. Em tese, essa tarefa poderia ser executada por agente de IA via API de algum LLM.

Todavia, essa opção revelou-se economicamente inviável para ser executada, pois cada decisão pode conter centenas ou milhares de palavras, gerando altos volumes de *tokens* por requisição. Na Open AI (modelo GPT, no plano *standard*), os preços variam de $0.05 (cinco centavos de dólar americano) para *input* e de $0.40 (quarenta centavos de dólar) por milhão de tokens[5]. Assim, a análise de dezenas de milhares de decisões poderia alcançar custo de milhares de dólares, inviável para pesquisador sem financiamento para a pesquisa.

Outra alternativa seria programar o sistema para ler arquivos em formato PDF e realizar buscas de *substrings* na *string* formada pelo texto integral de cada decisão. Essa abordagem, entretanto, revelou obstáculos significativos, uma vez que as *strings* extraídas não apresentam boa estrutura: contêm quebras de linha, inserções de

[3] COSTA, Alexandre e BENVINDO, Juliano, A Quem Interessa o Controle Concentrado De Constitucionalidade? - O Descompasso entre Teoria e Prática na Defesa dos Direitos Fundamentais (Who is Interested in the Centralized System of Judicial Review? - The Mismatch between Theory and Practice in the Protection of Basic Rights) (1 de abril de2014). Disponível no SSRN: https://ssrn.com/abstract=250954 ou http://dx.doi.org/10.2139/ssrn.2509541. Último acesso em 17/02/2026.

[4] URL: < https://github.com/AlexandreAraujoCosta?tab=repositories >.

[5] *Vide* tabela disponível em < https://platform.openai.com/docs/pricing?latest-pricing=standard >, acesso em 05/02/2026.

rodapé, numeração de páginas no meio de frases e, em alguns casos, mensagens sobre assinatura digital incluídas dentro dos períodos do texto. Tais fatores dificultam sobremaneira a utilização de *substrings* como critérios de pesquisa. Provavelmente seria necessário criar listas com milhares de substrings para identificar os fundamentos buscados em cada decisão. Além do tempo requerido para essa implementação, persistiria o risco de o sistema não alcançar um nível de precisão suficiente para assegurar confiabilidade adequada.

Uma terceira possibilidade, com potencial para superar os problemas enumerados acima, seria o uso da linguagem *RegEx* (*Regular Expressions*, ou expressões regulares, em português), para execução da tarefa de busca. Nesse caso, o desenvolvimento de algumas centenas de padrões textuais *RegEx* (ou *"patterns"*) poderia conferir ao sistema nível de confiabilidade mais robusto do que o obtido por simples listas de *substrings*.

2.2 Alteração do escopo

Ao examinar as informações disponibilizadas no código HTML das páginas de andamentos do portal do STF, observou-se a presença de dezenas de parâmetros com potencial de utilização em estudos de jurimetria. Por essa razão, alterou-se a abordagem inicialmente planejada: abandonou-se, ao menos neste trabalho, a intenção de programar a leitura automática dos PDFs das decisões e optou-se por construir código de automação voltado à raspagem e estruturação desses parâmetros, com marcadores que viabilizassem, por exemplo:

1) contagem de partes e de representantes processuais por polo processual;
2) a tabulação dos sujeitos processuais por polo (ativo, passivo e terceiros);
3) a contagem dos andamentos de cada processo;
4) o cálculo do tempo de tramitação dos processos;
5) a identificação dos atos praticados sob a direção da Presidência do Tribunal ou sob a direção do(a) Ministro(a) Relator(a);
6) a identificação das decisões proferidas, com *link* para acesso do documento;
7) detectar os recursos internos eventualmente interpostos;

8) a verificação de eventual reforma da decisão inicial;

9) o detalhamento, quando disponível na página de andamento, do dispositivo das decisões reformadoras;

10) a identificação dos julgamentos virtuais (datas do início e);

11) registrar os pedidos de vista eventualmente feitos durante os julgamentos colegiados, inclusive com o nome do(a) Ministro(a) que pediu vista;

12) identificar os pedidos de vista e de destaque, com indicação do(a) Ministro(a) responsável, quando disponível.

Ressalte-se que os andamentos processuais disponibilizados no portal do STF constituem, em termos práticos, um banco de dados com diversos parâmetros pesquisáveis. Existe, inclusive, ferramenta interna denominada Portal de Informações Gerenciais, que permite consultas parametrizadas e devolve dados tabulados; todavia, tal ferramenta não é disponibilizada universalmente, sequer ao público interno. Para fins de pesquisa, recorre-se ao portal Corte Aberta, que apresenta painéis estatísticos abrangendo diversas classes processuais[6] (figura 1).

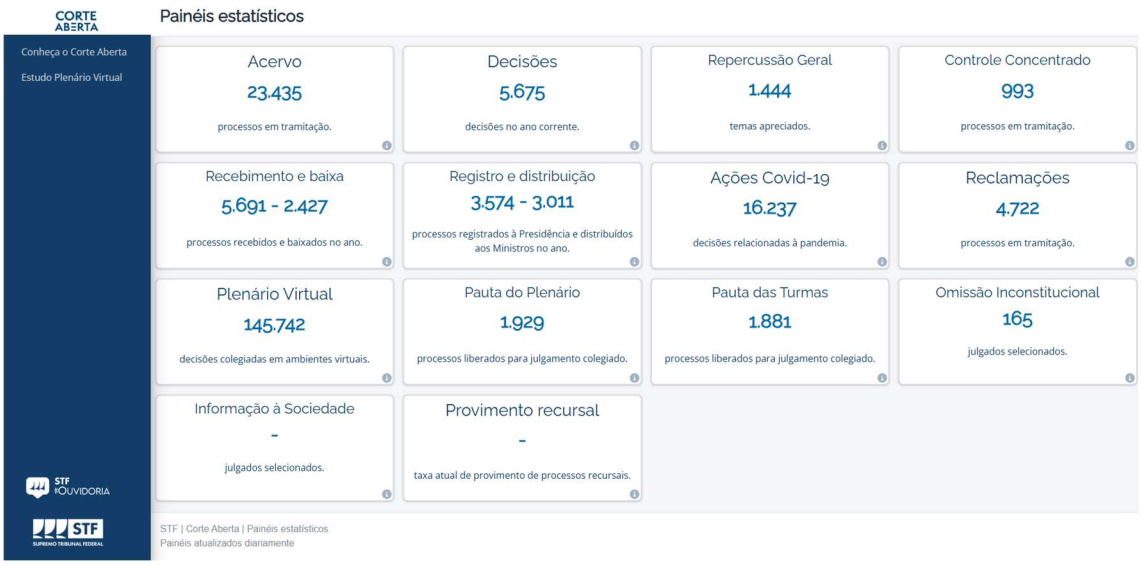

Figura 1

Tais painéis permitem ao usuário efetuar pesquisas por classe, por datas, por assunto e mais alguns outros parâmetros, podendo inclusive gerar planilhas. No

[6] URL < https://transparencia.stf.jus.br/extensions/corte_aberta/corte_aberta.html > Acesso em 24 de fevereiro de 2026.

entanto, as informações são compartimentadas por painéis, e, por conseguinte, os dados exportados em formato de planilha também mantêm essa compartimentação.

No painel “Acervo”, por exemplo, é possível levantar o acervo do Tribunal, incluindo informações sobre a existência de decisões nos processos ou de recursos internos pendentes. Todavia, a planilha gerada não indica as partes dos processos, que precisam ser consultadas na aba “Partes”, nem especifica qual recurso interno está pendente.

Dessa forma, o pesquisador precisa recorrer a diferentes painéis, exportar múltiplas planilhas e consolidar os dados manualmente — tarefa trabalhosa e sujeita a imprecisões. Além disso, nenhum dos painéis disponíveis apresenta todas as decisões proferidas nos processos acompanhadas dos respectivos links de acesso aos documentos.

O painel “Decisões” indica o tipo da última decisão proferida nos processos (se final, liminar, interlocutória ou em recurso interno), o andamento cadastrado (não conhecido, provido, não provido, improcedente *etc*) e a observação cadastrada no andamento (por exemplo:*“(...) dou parcial provimento ao recurso extraordinário (art. 932 do CPC) para reconhecer a contrariedade ao entendimento firmado na ADPF 1.060/DF e, como corolário, reformar o acórdão recorrido a fim de que o Tribunal Regional Federal da 2ª Região, afastada a prescrição, proceda a novo julgamento da apelação.”*).

A ausência de informações sobre todos os recursos internos interpostos pelas partes em cada processo inviabiliza o cálculo da taxa de recorribilidade das decisões da Corte ou do índice de reforma dos atos decisórios, sejam eles monocráticos ou colegiados.

Como já ressaltado, o Supremo Tribunal Federal não disponibiliza ao público — nem mesmo internamente — a leitura integral de seu banco de dados, medida justificada por razões de segurança em Tecnologia da Informação. Embora a disponibilização de dados no portal “Corte Aberta” represente um avanço relevante em termos de transparência, as informações oferecidas ainda possuem utilidade limitada para fins de pesquisa acadêmica.

Diante desse contexto, evidencia-se a relevância de ferramenta automatizada capaz de navegar pelo portal do STF, consultar páginas de andamentos, raspar dados

e armazená-los de forma organizada, permitindo que algoritmos especializados realizem cálculos e correlações para facilitar pesquisas empíricas.

Três diretrizes básicas orientaram a pesquisa e o desenvolvimento da aplicação: (i) uso de Python, por sua ampla adoção e curva de aprendizado relativamente suave; (ii) disponibilização gratuita do código em regime *open source*, permitindo visualização, modificação e redistribuição; e (iii) oferta de distribuição com interface gráfica integrada, tornando o software acessível a usuários sem domínio de Python e potencializando seu impacto como ferramenta de pesquisa.

2.3 Desenvolvimento da arquitetura da ferramenta

As primeiras versões do sistema — cujos códigos não foram publicados pelo autor — adotavam estrutura monolítica: um único módulo realizava a raspagem, o tratamento, a estruturação e a gravação dos dados tabulados em arquivo XLSX. Os testes demonstraram que essa decisão mascarava duas falhas críticas.

A primeira falha era a perda de informações na etapa de gravação, decorrente da limitação do Excel quanto ao armazenamento de caracteres por célula (32.767), já que há processos com milhares de partes ou centenas de andamentos, ultrapassando esse limite e provocando truncamento. Ademais, a arquitetura monolítica executava algoritmos de cálculo e análise imediatamente após a extração e antes da gravação: isso significava que a criação ou ajuste de algoritmos posteriormente exigiria nova raspagem, pois os cálculos não poderiam ser reaplicados com segurança ao mesmo *dataset* depois, já que parte das informações poderia ter se perdido o momento da gravação do XLSX. Ainda que fosse possível reprocessar a planilha, o resultado continuaria sujeito a incorreções em razão do truncamento.

A segunda falha, tão relevante quanto a primeira, era o tratamento insuficiente de erros de execução — especialmente os decorrentes de instabilidade de internet ou de ausência de resposta do servidor do STF —, capazes de interromper a execução do código. Como o arquivo XLSX era sobrescrito a cada iteração com acréscimo incremental, as interrupções tornavam necessário consolidar manualmente múltiplas planilhas ou desenvolver módulo adicional para unificar dados gravados em cada arquivo gravado quando a execução do código era interrompida.

Esses problemas exigiram nova arquitetura baseada em duas diretrizes: (i) separação de tarefas por módulos e (ii) gravação dos dados extraídos em formatos que evitassem perda de informações. Com efeito, um único módulo recebeu a atribuição de executar a raspagem de dados e gravar um único arquivo contendo os dados de cada processo consultado e um segundo módulo foi desenvolvido para executar a tarefa de – a partir da leitura dos arquivos gerados pelo módulo anterior – executar cálculos, analisar andamentos e decisões e estruturar novas informações, para, ao final, tabular todos os dados e gravá-los em um arquivo com formato XLSX.

Para gravação dos dados extraídos por processo, adotaram-se os formatos TXT e JSON (*JavaScript Object Notation*). O JSON é leve, amplamente utilizado para troca de dados (especialmente em aplicações web e APIs), reveste-se de boa legibilidade humana e é facilmente interpretado por máquinas. Além disso, a estratégia de gravar arquivos individuais por processo tende a reduzir a pressão sobre a memória RAM durante execuções prolongadas, pois, ao salvar dados em disco, libera-se memória que pode ser reutilizada, evitando crescimento incremental do consumo ao longo de milhares de iterações, que na arquitetura anterior era provocado pela adição constante de novos valores ao dicionário no qual eram armazenados, a cada nova iteração, os dados extraídos e não podia ser limpo, sob pena de inviabilizar a gravação da planilha ao final ciclo do laço de repetição.

A gravação final em XLSX permanece imprescindível por ser formato amigável à maioria dos pesquisadores e útil a ferramentas como Tableau e Power BI, embora deve-se reconhecer que bancos de dados SQL eliminariam, em tese, limitações do Excel. Todavia, essa solução exigiria conhecimentos adicionais (SQL), reduzindo a acessibilidade. Por isso, manteve-se a saída final em planilha (apesar da limitação de caracteres por célula) e optou-se por conservar integralmente, em formato alternativo (JSON), a integralidade dos dados consolidados, o que permite recuperação de informações que venham a ser truncadas durante a gravação da planilha XLSX e, sobretudo, auditoria dos cálculos executados pelos algoritmos.

Assim, em caso de aprimoramento e expansão do código, com o desenvolvimento de um módulo específico — baseado em um agente de IA ou em linguagem *RegEx* — para identificar os fundamentos das decisões, não haveria necessidade de executar nova raspagem de dados. Nesse cenário, bastaria

encaminhar os dados já extraídos anteriormente ao novo módulo de processamento, implementando-se uma nova saída tabulada com o acréscimo das informações sobre os fundamentos das decisões.

Sanadas as deficiências relacionadas à perda de informação e à consolidação, revelou-se terceiro obstáculo: a utilização do script poderia restringir-se a pesquisadores com conhecimento mínimo de Python, dado que a execução exigia instalação de interpretador, *IDE* e edição manual de parâmetros no código (intervalos/listas de processos, diretórios de gravação/leitura de arquivos, inclusive os de logs etc.). Além disso, alterações acidentais em linhas indevidas poderiam inutilizar o script, e a depuração demandaria esforço incompatível com o público-alvo.

A superação dessa barreira exigiu três etapas: (i) criação de *GUI* (*Graphical User Interface*) com campos de parametrização (classes, intervalos/listas, diretórios etc.); (ii) integração da *GUI* com os módulos, encapsulando cada módulo em funções acionáveis pela interface; e (iii) empacotamento da aplicação, com dependências, em arquivo autoexecutável, capaz de rodar independentemente de interpretador e IDE instalados no computador do usuário. Assim, obtém-se distribuição acessível a usuários sem conhecimento prévio de programação.

2.4 Arquitetura final

Considerando as implicações discutidas, a arquitetura final foi definida por: (i) separação de tarefas por módulos; (ii) gravação de dados em JSON e XLSX; e (iii) geração de logs (provisórios, definitivos e de erro), com barra de progresso da extração. Foram desenvolvidos cinco componentes, cada qual voltado à execução de tarefa específica, conforme descrito a seguir.

2.4.1 Caixa de ferramentas (biblioteca de suporte)

Tem atribuição exclusiva armazenar estruturas de dados e funções que são chamadas pelos demais, evitando-se a repetição demasiada de linhas de código nos demais módulos.

2.4.2 Módulo de extração

Responsável por navegar pelo portal do STF, executar a raspagem, estruturar as informações coletadas, gravar arquivos individuais por processo e gerar logs da extração.

2.4.3 Módulo de enriquecimento de dados

Responsável por ler os arquivos gerados pelo módulo de extração, extrair a numeração única de cada processo, consultar a API pública do CNJ (*DataJud*) em busca do processo correspondente e, a partir da resposta, estruturar e gravar arquivos individuais enriquecidos por processo.

2.4.5 Módulo de consolidação dos dados

Responsável por consolidar os dados raspados, lendo os arquivos JSON gerados pelos módulos de extração e enriquecimento.

Os dados são estruturados em grande dicionário, percorrido por algoritmos que: calculam o tempo de tramitação; identificam atos relevantes sob a Presidência e sob a relatoria; indicam decisões e observações registradas nos andamentos; detectam reforma de decisões durante recursos internos; verificam julgamento virtual; e registram pedidos de vista e de destaque, com indicação do(a) Ministro(a) responsável quando disponível. Como saída, o módulo gera um dicionário consolidado em JSON e uma planilha tabulada em XLSX.

2.4.6 Interface gráfica do usuário

A *GUI* abre uma janela com três abas. Na primeira, o usuário escolhe a classe processual, define pesquisa por intervalo ou por lista, seleciona diretórios de gravação dos dados e logs, ajusta pausas (quando aplicável) e inicia a extração. Na segunda, o usuário seleciona pastas de dados extraídos do STF e de saída do enriquecimento, informa a chave de acesso à API e inicia a consulta ao *DataJud*. Na terceira, o usuário define diretórios de entrada (STF e CNJ) e de saída, nomeia os arquivos XLSX e JSON finais e inicia a consolidação. Em cada aba, um painel de status exibe, em tempo real, informações sobre execução, inclusive sobre os erros tratados, ou não, que venham a ocorrer durante a execução do programa. Ao final da consolidação, o

painel do módulo correspondente exibe lista de processos cuja tramitação não foi encerrada, o que pode demandar, conforme o escopo de pesquisa, nova extração futura.

2.5. Distribuição autoexecutável

O *software* foi desenvolvido e empacotado para execução no sistema operacional Windows. Assim, a distribuição autoexecutável roda apenas em Windows e exige os seguintes requisitos mínimos e recomendados (Quadro 1).

Mínimos	Recomendado
– Windows 10 64 bits – CPU dual-core 2 GHz – 4 GB de RAM – 1 GB livre em disco (incluindo TEMP) – Google Chrome instalado + ChromeDriver compatível – Acesso à internet para portal STF e API DataJud	– Windows 10/11 64 bits – CPU quad-core – 8 GB de RAM – 5 GB livres em disco para dados e logs

Quadro 1

A disponibilização de distribuição para iOS demandaria reescrita em outra linguagem e enfrentaria restrições técnicas e de política do ecossistema Apple (assinatura obrigatória de código e limitações a automações de navegador), tornando a pretensão inviável no horizonte temporal de uma pesquisa de mestrado.

Por não ser usuário Linux, o desenvolvedor não realizou empacotamento para esse ambiente; todavia, os arquivos PY da interface e dos módulos estão publicados no repositório do desenvolvedor, permitindo que terceiros realizem empacotamento com *PyInstaller* em ambiente Linux, sem a necessidade de alteração de código.

3. ESTRUTURAÇÃO DOS DADOS EXTRAÍDOS PELOS MÓDULOS DE EXTRAÇÃO E DE ENRIQUECIMENTO

Conforme exposto nas justificativas sobre a arquitetura adotada, a gravação de arquivos individuais por processo mostrou-se estratégia necessária para enfrentar a perda de dados decorrente da estrutura monolítica inicial, na qual as informações eram gravadas apenas em planilhas XLSX após a extração e o processamento.

Inicialmente, adotou-se o formato TXT, pois, além de preservar a integralidade das informações raspadas, a estrutura do arquivo se revelou amigável à leitura humana, com leiaute semelhante ao exibido pelo navegador (figuras 2 e 3)

```
ARE 1401627/BA - BAHIA

Classe: ARE
Número: 1401627
Número único: 0000901-84.2021.8.05.9000
Órgão de origem: TJBA - 1ª TURMA RECURSAL
Origem: BA - BAHIA
Modo de tramitação: PROCESSO ELETRÔNICO
Publicidade: PÚBLICO
Critérios de prioridade: Nenhum
Relator: MINISTRO PRESIDENTE
Redator para o acórdão: Não há
Relator do último incidente: Não há recurso interno
Último incidente: Não há
Assuntos:
        DIREITO PROCESSUAL CIVIL E DO TRABALHO | Formação, Suspensão e Extinção do Processo | Extinção do Processo Sem Resolução de Mérito | Inépcia
da Inicial

Partes:
RECTE.(S): EDMILSON GAMA SANTIAGO
ADV.(A/S): EDMILSON GAMA SANTIAGO (66494/BA)
RECDO.(A/S): ESTADO DA BAHIA
ADV.(A/S): PROCURADOR-GERAL DO ESTADO DA BAHIA

Andamentos:
12/09/2022 - Protocolado # PROCESSO PROTOCOLADO VIA SISTEMA STF-TRIBUNAIS. *  $  ->
13/09/2022 - Autuado #  *  $  ->
13/09/2022 - Registrado à Presidência #  *  $  ->
13/09/2022 - Conclusos à Presidência #  *  $  ->
16/09/2022 - Determinada a devolução # É incabível o agravo dirigido ao STF contra decisão que nega seguimento ao recurso extraordinário com base na
sistemática de repercussão geral (art. 1.030, § 2º, do CPC/2015). * PRESIDÊNCIA $ Despacho -> downloadPeca.asp?id=15353586080&ext=.pdf
16/09/2022 - Remessa externa dos autos, Guia nº # Guia: 29909/2022 - TJBA - BA - 6ª TURMA RECURSAL *  $ Termo de remessa -> downloadPeca.asp?id=
15353586456&ext=.pdf
19/09/2022 - Publicação, DJE # Divulgado em 16/09/2022 *  $  ->
26/09/2022 - Autos disponibilizados à origem #  *  $  ->
04/10/2022 - Processo recebido na origem # TJBA - 6ª TURMA RECURSAL *  $  ->

Links das decisões:
16/09/2022 - Despacho de devolução 1 (despacho) # https://portal.stf.jus.br/processos/downloadPeca.asp?id=15353586080&ext=.pdf

Link's dos outros documentos disponíveis:
16/09/2022 - Despacho # https://portal.stf.jus.br/processos/downloadPeca.asp?id=15353586080&ext=.pdf
16/09/2022 - Termo de remessa # https://portal.stf.jus.br/processos/downloadPeca.asp?id=15353586456&ext=.pdf

Link para consulta:
        https://portal.stf.jus.br/processos/detalhe.asp?incidente=6479206


Dia e hora da extração: 20/10/25 14:11:44
```

Figura 2

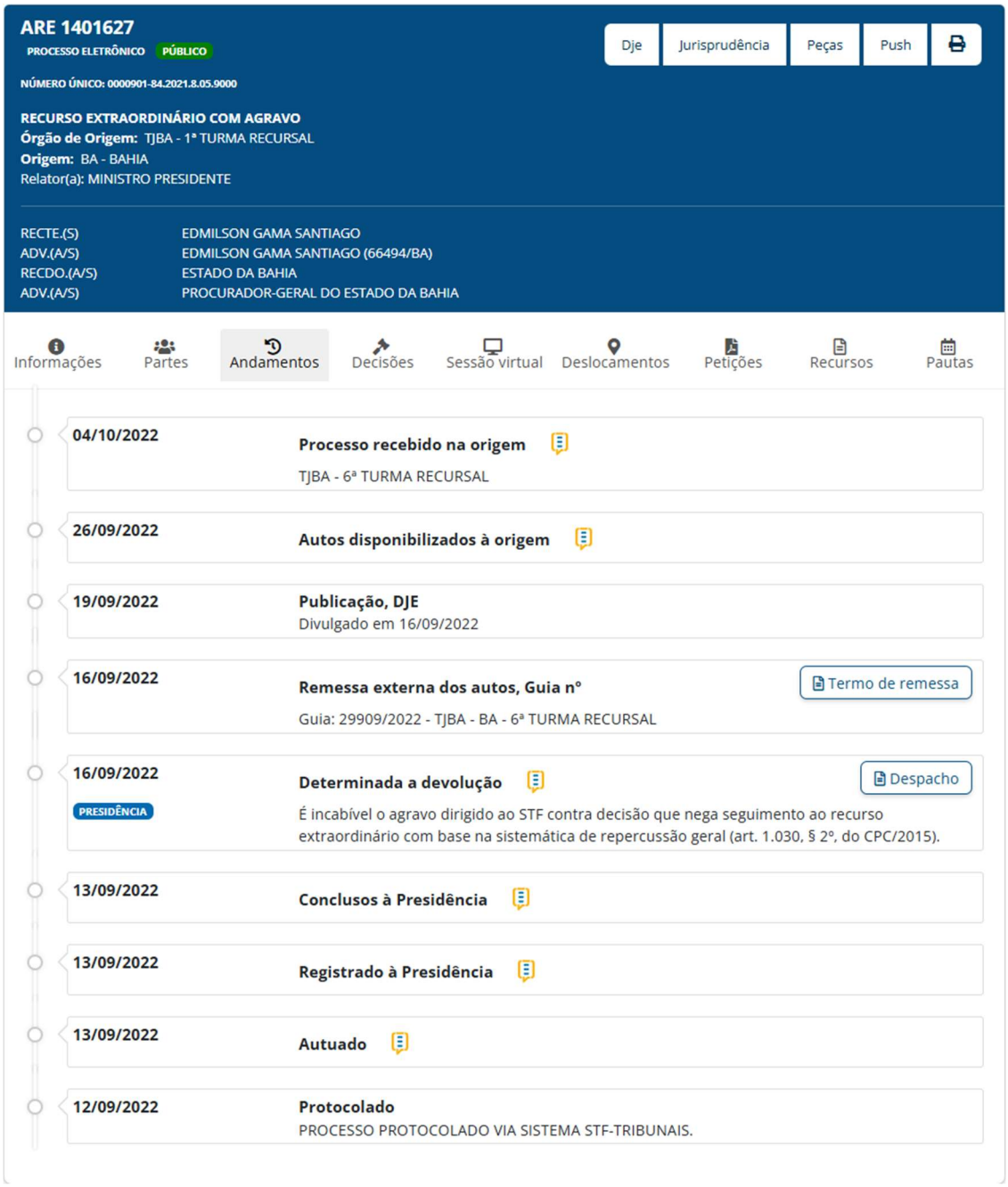

Figura 3

Não obstante, embora visualmente compreensíveis, os dados em TXT consistem essencialmente em uma coleção de caracteres (*strings*) que, embora legível por máquinas, carece de estrutura adequada para viabilizar sua leitura por algoritmos robustos de cálculo e análise.

Nesse contexto, merece destaque a advertência de Fabiano Hartmann Peixoto e Debora Bonat[7] quanto à relevância da organização de *datasets* como elemento sensível para alcance e impacto de sistemas de inteligência artificial, cuja lógica inspirou a decisão de aprimorar também a forma de armazenamento dos dados neste projeto:

> *"Este grau de sensibilidade na formação de datasets é tamanho, que no DR.IA desenvolvemos um conceito ampliado da própria IA, englobando também o processo de formação de datasets. O nosso conceito "Lego" de IA vai além de uma visão puramente algorítmica da IA. Assim, temos a inteligência artificial como sistemas que buscam a reprodução parcial da atividade cognitiva realizada por seres humanos com o arranjo indispensável de três elementos: dataset, combinação algorítmica e resultados aferíveis. Da mesma forma como o brinquedo infantil, só haverá sentido com as peças devidamente combinadas. Portanto, os três elementos são essenciais para que uma IA robusta seja compatível com as diretrizes de confiabilidade e respeito, portanto, de adequação a um tratamento jurídico protetivo aos direitos fundamentais."*

Embora o sistema aqui descrito constitua, essencialmente, uma automação de formação de *dataset* — e não uma IA em sentido estrito —, a preocupação metodológica com a estruturação dos dados orientou a adoção de um segundo formato de armazenamento, além do TXT, conforme diretriz fundamental do conceito "Lego".

Por essa razão, adotou-se o formato JSON (*JavaScript Object Notation*), que é leve, amplamente usado em aplicações web e APIs, de escrita e leitura acessíveis a humanos e facilmente interpretável por máquinas.

O JSON permite armazenar as informações em estrutura do tipo dicionário, composta por chaves cujos valores podem ser acessados separadamente, o que favorece a construção de rotinas analíticas e reduz fragilidades típicas de formatos menos estruturados.

---

[7] PEIXOTO, Fabiano Hartmann; BONAT, Debora. GPTs e Direito: impactos prováveis das IAs generativas nas atividades jurídicas brasileiras . Seqüência Estudos Jurídicos e Políticos, Florianópolis, v. 44, n. 93, p. 1–31, 2023. DOI: 10.5007/2177-7055.2023.e94238. Disponível em: https://periodicos.ufsc.br/index.php/sequencia/article/view/94238. Acesso em: 7 fev. 2026.

Sob esse aspecto, aliás, a Universidade de Yale [8] aponta as vantagens do JSON para intercâmbio com modelos de linguagem (LLMs), por preservar hierarquia e tipos de dados, evitar problemas de *parsing* e comportar múltiplos conjuntos relacionados em um único arquivo.

No sistema descrito, o dicionário JSON do módulo de extração foi estruturado com 26 chaves, cujos valores abrangem *strings*, inteiros e listas.

Uma dessas chaves armazena os andamentos processuais. Cada andamento exibido pelo portal é decomposto em seus elementos constitutivos (data, título, detalhamento, órgão julgador, descrição do documento e *link*) e convertido em *string* com separadores textuais padronizados.

Foram escolhidos, de modo arbitrário, os separadores "-", "#", "*", "$" e "->", sempre com espaço antes e depois de cada um desses caracteres, para manter legibilidade e facilitar a extração posterior de *substrings* específicas com base nesses marcadores, contendo os textos obtidos a partir da decomposição dos elementos constitutivos do código HTML em relação às seguintes informações: data, título do andamento, detalhamento do andamento, órgão julgador, descrição do documento disponibilizado no andamento e *link* de acesso ao documento (figura 4):

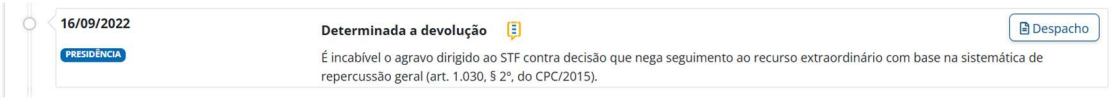

Figura 4

Com efeito, o andamento da figura exemplificativa acima é convertido em item da lista da chave do dicionário à qual se atribuiu a responsabilidade de guardar a lista de andamentos do processo consultado, como a seguinte *string*: "*16/09/2022 - Determinada a devolução # É incabível o agravo dirigido ao STF contra decisão que nega seguimento ao recurso extraordinário com base na sistemática de repercussão geral (art. 1.030, § 2º, do CPC/2015). * PRESIDÊNCIA $ Despacho -> downloadPeca.asp?id=15353586080&ext=.pdf*".

[8] URL: < https://ai.yale.edu/yales-ai-tools-and-resources/clarity-platform/formatting-files > , acesso em 07/02/2026.

Como o HTML de cada andamento, em regra, disponibiliza apenas o sufixo do link do documento, os algoritmos responsáveis por construir as listas de documentos completam o endereço com o prefixo do portal do STF, assegurando links funcionais na estrutura final (*https://portal.stf.jus.br/processos/*), assegurando a existência de *links* funcionais na estrutura final (figura 5):

Figura 5

A seguir, apresenta-se o conjunto de chaves do JSON de saída do módulo de extração, com indicação do tipo esperado (padronização editorial aplicada):

- “classe”: ***string***;
- “numero”: ***número inteiro***;
- “incidente”: ***string***;
- “numero_unico”: ***string***;
- “orgao_de_origem”: ***string***;
- “uf_de_origem”: ***string***;
- “modo_tramitacao”: ***string***;
- “publicidade”: ***string***;
- “prioridades”: ***lista***;
- “paradigma_rep_geral”: ***string***;
- “relator”: ***string***;
- “redator_acordao”: ***string***;
- “ultimo_incidente”: ***string;***
- “relator_ultimo_incidente”: ***string***;
- “assuntos”: ***lista***;
- “partes”: ***lista***;
- “p_ativo”: ***lista***;
- “rep_proc_ativo”: ***lista***;
- “p_passivo”: ***lista***;
- “rep_proc_passivo”: ***lista***;

- “terceiros”: ***lista***;
- “rep_proc_terc”: ***lista***;
- “andamentos”: ***lista***;
- “docs_decisoes”: ***lista***;
- “outros_docs”: ***lista***; e
- “data_extração”: ***string***.

Durante a construção do módulo de extração, buscou-se ampliar sua aplicabilidade a múltiplas classes processuais. Para isso, a “caixa de ferramentas” (ModuloSTFSelenium.py) contém dicionário que mapeia: (i) nomenclatura e siglas de classes processuais; (ii) terminologia dos sujeitos processuais por polo e de intervenientes (tratados de forma simplificada como “terceiros”); e (iii) terminologia e siglas de procuradores.

Esse arranjo viabiliza separar a lista geral de partes e procuradores (armazenada em partes) em listas específicas por polo (p_ativo, p_passivo, terceiros) e por representação (rep_proc_ativo, rep_proc_passivo, rep_proc_terc), independentemente da classe processual.

A ampliação do uso potencial do módulo de raspagem exigiu, contudo, um “pedágio” metodológico: uma simplificação que, em alguma medida, prejudica o uso estritamente técnico de categorias jurídico-processuais.

Exemplo ilustrativo: em ações do controle concentrado (ação direta de inconstitucionalidade, ação declaratória de constitucionalidade, ação direta de constitucionalidade por omissão e arguição de descumprimento de preceitos fundamentais) sabe-se que não há “partes” no sentido tradicional de lide, mas, para fins de tabulação, o código classifica o legitimado ativo como polo ativo e categoriza o ente de origem do ato normativo e os “*amici curiae”* como “terceiros”, o que não corresponde à melhor técnica processual, mas preserva viabilidade de análise empírica por padronização.

Vê-se, a partir do exemplo acima, que cada classe processual pode conter especificidades próprias e tratá-las individualmente, para ampliar a potencialidade de utilização do *software*, além de constituir atividade trabalhosa, tornaria o código mais

complexo, potencializando as chances resultados imprecisos e dificultando a sua manutenção.

A simplificação, desde que declarada, contorna barreiras de modelagem e não inviabiliza pesquisas, na medida em que o objetivo é estruturar dados para análise quantitativa e não reproduzir, com exatidão dogmática, todas as categorias do direito processual.

4. TRATAMENTO DE ERROS

A lógica do módulo de raspagem é relativamente simples: abre-se uma instância do navegador, acessa-se o portal do STF, preenchem-se campos de pesquisa (classe e número do processo), aciona-se a pesquisa e, na página de andamentos, o código identifica os elementos a serem extraídos (sujeitos processuais, relatoria, numeração única etc.), além de capturar assunto e demais dados relevantes; em seguida, repete-se o ciclo para o próximo número do intervalo ou da lista definida

4.1 Alternância entre as classes ARE e RE e número processual não encontrado

A numeração da maioria das classes processuais é sequencial; contudo, ARE e RE compartilham sequência numérica, de modo que um número pode corresponder a ARE ou a RE. Assim, quando a consulta por determinado número não retorna página contendo os dados esperados, o sistema alterna a classe (ARE↔RE) e repete a consulta.

Mesmo com essa alternância, há números que não correspondem a processo em nenhuma das duas classes, hipótese em que, se não tratada, pode interromper prematuramente o laço de repetição. Por isso, desde as primeiras versões, tornou-se necessário tratar tais erros para preservar estabilidade da extração.

Inicialmente, quando a arquitetura ainda era monolítica e gravava planilhas incrementais, com sobrescrita do arquivo anterior, optou-se por ignorar processos inexistentes e avançar para a próxima iteração. Com a evolução arquitetural, implementou-se geração sistemática de arquivos de log em TXT ao final de cada ciclo de extração.

Os logs registram: Os logs registram: (i) quantidades de consultas solicitadas, de consultas bem-sucedidas e de processos inexistentes; (ii) relação nominal dos processos extraídos com sucesso; (iii) números sem correspondência processual; (iv) duração total da execução; (v) quantidade e duração total das pausas; (vi) tempo médio por consulta; (vii) indicador de progresso; e (viii) descrição de erro, quando houver.

Dessa forma, caso a execução seja interrompida abruptamente, o usuário pode verificar o último processo extraído com sucesso e retomar a extração a partir do número subsequente, reduzindo perda de tempo e aumentando reprodutibilidade operacional.

4.2 Bloqueios pelo servidor e quedas de internet

Em condições ideais, o tempo médio por consulta é inferior a 10 segundos, o que permitiria extração de grande volume diário. Entretanto, um alto número de requisições automatizadas em intervalo curto aciona mecanismos de proteção do servidor (mitigação de estrangulamento, sobrecarga e ataques), levando a recusas de requisições. Isso gera erros típicos de extração (elementos não encontrados no HTML), capazes de encerrar a execução se não forem tratados.

O tratamento não pode se limitar a repetir indefinidamente a mesma consulta, sob pena de sucessivas negativas. Por isso, implementou-se um mecanismo antibloqueio baseado em pausas inteligentes.

Primeiro, criou-se contador de consultas e uma rotina de pausa a cada ciclo de 50 consultas, por intervalo aleatório entre 150 e 240 segundos. Isso aumentou estabilidade, permitindo centenas de consultas antes de bloqueios, mas não eliminou o problema, pois o ponto de bloqueio variava entre execuções.

Em seguida, acrescentou-se algoritmo adicional que implementa pausas aleatórias (entre 20 e 59 segundos) a cada ciclo aleatório de 10 a 30 consultas, elevando a tolerância do servidor às requisições automatizadas. Ainda assim, bloqueios continuaram a ocorrer com alguma frequência, causando interrupção da do código.

Então, foi incluído mecanismo opcional, acionável pelo usuário, que adiciona pausa curta (entre 2 e 8 segundos) ao final de cada extração bem-sucedida. Essa opção reduz bloqueios em certos cenários, mas aumenta significativamente o tempo médio de consulta e não garante que o servidor deixe de detectar tráfego automatizado.

O maior inconveniente decorrente da detecção de tráfego automatizado é que a extração de dados passa a exigir supervisão constante, reduzindo de forma significativa a principal vantagem da automação: a economia de tempo e esforço humano. Caso contrário, corre-se o risco de perda substancial de produtividade. Por exemplo, se o usuário programasse a extração de quatro mil processos e apenas verificasse a execução após três horas, poderia encontrar a seguinte situação: um bloqueio ocorrido nos primeiros 30 minutos teria causado a interrupção abrupta do código, deixando a máquina inativa pelas duas horas e meia seguintes.

A experiência empírica indicou maior tolerância do servidor durante madrugadas e fins de semana e menor tolerância em horários de pico (dias úteis, períodos diurnos e início da noite). Em vez de ajustar toda a lógica com base nessa janela temporal, optou-se por criar tratamento de falhas para impedir que bloqueios interrompessem a execução prematuramente.

Assim, criou-se algoritmo que, diante de recusa do servidor ou queda do sinal de internet, fecha o navegador e interrompe requisições por intervalo aleatório entre 45 e 60 minutos; após o período de pausa, inicia nova instância e retoma a consulta a partir do processo cuja requisição havia sido negada.

Para garantir que, nas hipóteses de o servidor do STF sair do ar por longo prazo ou de o usuário do programa ficar sem sinal de internet por tempo indefinido, outra regra foi implementada: a execução da extração é interrompida de modo seguro, gerando o respectivo log de erro quando ocorrer a quarta pausa consecutiva por bloqueio do servidor ou queda de internet, sem que nenhuma extração seja feita entre essas pausas.

As melhorias ora relatadas deixaram o sistema bem mais robusto, a ponto de chegar a ser executado por 95h:46m:16s entre os dias 16 e 20/1/2026, realizando a extração de 20.002 (vinte mil e dois) processos neste intervalo, tendo executado 1.214 (mil duzentas e quatorze pausas), cuja duração somou o tempo total de 31h:24m:33s.

Observa-se, assim, que aproximadamente um terço do tempo total de execução é consumido por pausas antibloqueio, elevando o tempo médio prático por processo para algo próximo de 20 segundos, em contraste com o potencial ideal de cerca de 8 segundos. Apesar de a parcela de tempo consumida não ser desprezível,

o custo foi considerado aceitável em face do ganho de estabilidade e redução da necessidade de supervisão humana constante.

4.3 Fechamento e reinício periódicos do navegador

Em execuções prolongadas, podem surgir erros decorrentes do acúmulo de recursos (consumo progressivo de memória/CPU) ou instabilidades do *WebDriver* em sessões longas. Em vez de tratar a diversidade de erros após sua ocorrência (solução mais complexa), adotou-se prevenção: a cada ciclo de 750 consultas, o navegador é encerrado e reiniciado, retomando-se as consultas a partir do processo seguinte ao último extraído com sucesso antes do encerramento da sessão.

4.4 Erros na consulta à *API DataJud*

O módulo de enriquecimento lê, em cada arquivo JSON extraído do portal do STF, a numeração única do processo, observando a estrutura prevista na Resolução CNJ nº 65/2008 (NNNNNNN-DD.AAAA.J.TR.OOOO), nos termos da Resolução CNJ nº 65, de 16/12/2008:

> *"Art. 1º Fica instituída a numeração única de processos no âmbito do Poder Judiciário, observada a estrutura NNNNNNN-DD.AAAA.J.TR.OOOO, composta de 6 (seis) campos obrigatórios, nos termos da tabela padronizada constante dos Anexos I a VII desta Resolução.*
>
> *(...)*
>
> *§ 4º O campo (J), com 1 (um) dígito, identifica o órgão ou segmento do Poder Judiciário, observada a seguinte correspondência:*
>
> *I – Supremo Tribunal Federal: 1 (um);*
>
> *II – Conselho Nacional de Justiça: 2 (dois);*
>
> *III – Superior Tribunal de Justiça: 3 (três);*
>
> *IV - Justiça Federal: 4 (quatro);*
>
> *V - Justiça do Trabalho: 5 (cinco);*
>
> *VI - Justiça Eleitoral: 6 (seis);*
>
> *VII - Justiça Militar da União: 7 (sete);*
>
> *VIII - Justiça dos Estados e do Distrito Federal e Territórios: 8 (oito);*
>
> *IX - Justiça Militar Estadual: 9 (nove).*

> *§ 5º O campo (TR), com 2 (dois) dígitos, identifica o tribunal do respectivo segmento do Poder Judiciário e, na Justiça Militar da União, a Circunscrição Judiciária, observando-se:*
>
> *I – nos processos originários do Supremo Tribunal Federal, do Conselho Nacional de Justiça, do Superior Tribunal de Justiça, do Tribunal Superior do Trabalho, do Tribunal Superior Eleitoral e do Superior Tribunal Militar, o campo (TR) deve ser preenchido com zero;*
>
> *II – nos processos originários do Conselho da Justiça Federal e do Conselho Superior da Justiça do Trabalho, o campo (TR) deve ser preenchido com o número 90 (noventa);*
>
> *III – nos processos da Justiça Federal, os Tribunais Regionais Federais devem ser identificados no campo (TR) pelos números de 01 a 06, observadas as respectivas regiões; (Redação dada pela Resolução nº 477, de 10.10.2022)*
>
> *IV – nos processos da Justiça do Trabalho, os Tribunais Regionais do Trabalho devem ser identificados no campo (TR) pelos números 01 a 24, observadas as respectivas regiões;*
>
> *V – nos processos da Justiça Eleitoral, os Tribunais Regionais Eleitorais devem ser identificados no campo (TR) pelos números 01 a 27, observados os Estados da Federação, em ordem alfabética;*
>
> *VI – nos processos da Justiça Militar da União, as Circunscrições Judiciárias Militares devem ser identificadas no campo (TR) pelos números 01 a 12, observada a subdivisão vigente;*
>
> *VII – nos processos da Justiça dos Estados e do Distrito Federal e Territórios, os Tribunais de Justiça devem ser identificados no campo (TR) pelos números 01 a 27, observados os Estados da Federação e o Distrito Federal, em ordem alfabética;*
>
> *VIII – nos processos da Justiça Militar Estadual, os Tribunais Militares dos Estados de Minas Gerais, Rio Grande do Sul e São Paulo devem ser identificados no campo (TR) pelos números 13, 21 e 26, respectivamente, cumprida a ordem alfabética de que tratam os incisos V e VII;"*

Por meio dessa numeração, identifica-se ramo da justiça e tribunal de origem, o que permite direcionar a requisição ao *endpoint* apropriado.

Os dados retornados são gravados em um arquivo JSON por processo, em estrutura de dicionário com 13 chaves, incluindo informações como classe na origem, sistema, tribunal, grau, órgão julgador, nível de sigilo, assuntos, andamentos e *timestamp* de extração:

```
        {
          "classe_stf": "RE",
          "numero_stf": 1527076,
          "processo": "5070002-37.2020.4.02.5101",
          "fonte": "api_publica_trf2",
          "classe": "Apelação Cível",
          "sistema": "EPROC",
          "modo_tramitacao": "Eletrônico",
          "tribunal": "TRF2",
          "grau": "G2",
          "orgao_julgador": "Gabinete da Vice-Presidência",
          "nivel_sigilo": 0,
          "assuntos": "Aposentadoria Especial (Art. 57/8)",
          "andamentos": [
            "Distribuição - sorteio * 2021-11-05T16:16:06.000Z",
            "Redistribuição - recusa de prevenção/dependência * 2022-05-16T12:48:24.000Z",
            "Publicação - Acórdão * 2024-01-31T23:51:02.000Z",
            "Petição - Embargos de declaração * 2024-02-28T16:23:31.000Z",
            "Petição - Recurso especial * 2024-07-18T12:11:14.000Z",
            "Redistribuição - recusa de prevenção/dependência * 2024-08-21T15:12:44.000Z",
            "Remessa - em grau de recurso * 2024-08-26T17:35:13.000Z",
            "Recurso Extraordinário com repercussão geral * 2025-01-08T12:53:33.000Z",
            "Recebimento * 2022-05-16T12:48:24.000Z",
            "Sentença desconstituída * 2024-01-25T23:12:34.000Z",
            "Não-Acolhimento de Embargos de Declaração * 2024-05-30T19:17:08.000Z",
            "Recebimento * 2024-08-21T15:12:44.000Z",
            "Recebimento * 2024-12-30T13:14:44.000Z"
          ],
          "data_e_hora_da_extração": "09/02/2026 - 01:19:45"
```


A existência de um **par de arquivos** por processo (STF + *DataJud*) é **fundamental** para a consolidação posterior. Por isso, mesmo quando não há retorno útil da API, o sistema deve gerar arquivo correspondente, sob pena de inviabilizar a montagem da planilha final.

Há casos em que não existe *endpoint* (por exemplo, STF, CJF, STJ, TNU e Auditorias Militares); nessas hipóteses, o módulo registra no dicionário a informação de indisponibilidade de API para a origem e mantém, ao menos, os identificadores básicos do processo (classe STF, número STF, numeração única).

Em outros casos, o número único não está disponível na página de andamentos do STF, o que inviabiliza a consulta à API; nessa situação, o sistema armazena nas chaves do dicionário a informação *"consulta impossivel, numero unico nao disponível"*.

Se o processo tramitou sob sigilo na origem, a API pode retornar dicionário vazio; nesse caso, o sistema registra "não há dados disponíveis" nas chaves pertinentes do arquivo de enriquecimento.

Há, ainda, hipótese de erro de conexão com a API: o módulo grava "erro de conexão com a API" nas chaves contidas no arquivo JSON, mas os dados podem existir. Para lidar com isso, o sistema implementa *loop* de tentativas sucessivas: após consultar todos os processos, relê os arquivos e repete requisições para aqueles marcados com erro de conexão, sobrescrevendo o arquivo primitivo quando houver retorno útil ou confirmação de ausência de dados. O *loop* se encerra quando não existirem mais arquivos com indicação de erro de conexão.

4.5 Advertências ao usuário pela interface gráfica

A *GUI* foi concebida para ser simples, limpa e intuitiva, permitindo parametrizar extração, enriquecimento e consolidação. Para assegurar uso correto, cada módulo exige parâmetros obrigatórios, e a interface exibe janelas *pop-up* quando o usuário tenta executar o módulo sem preenchimento adequado dos campos (figuras 5, 6 e 7).

No módulo de extração, por exemplo, exige-se: (i) classe processual; (ii) definição do modo de pesquisa (intervalo ou lista); (iii) números inicial/final, quando intervalo; (iv) lista separada por vírgulas, quando lista; e (v) diretórios de gravação de dados e logs. No módulo de consolidação, exige-se seleção das pastas onde estão gravados os dados extraídos e enriquecidos, bem como diretório de saída.

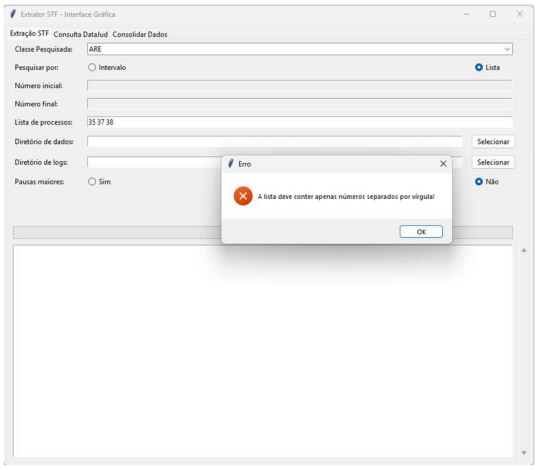

Figura 5

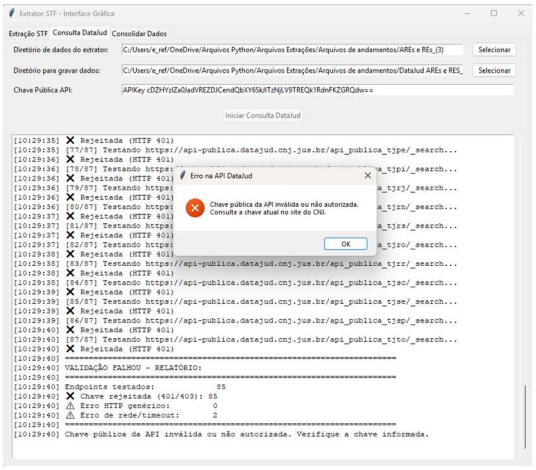

Figura 6

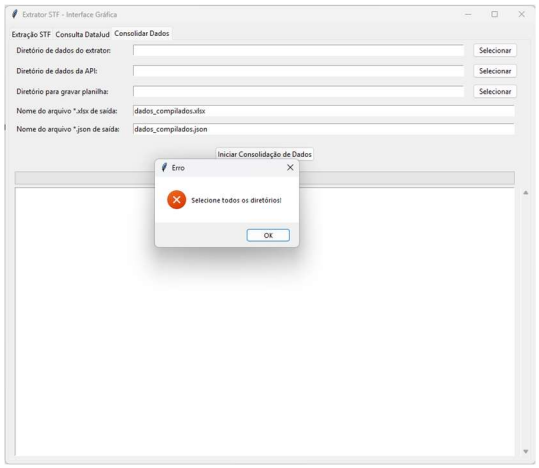

Figura 7

A chave de acesso aos *endpoints* do *DataJud* é pré-carregada com a vigente no momento do desenvolvimento mas o código testa sua validade antes de iniciar requisições; se inválida, a interface orienta o usuário a obter chave atual no site do CNJ (figura 7).

4.6 Erros não tratados

Apesar da robustez alcançada, existem erros não tratados ou tratados de modo insuficiente. Um exemplo é o fechamento intencional ou acidental da instância do *WebDriver*, o que interrompe inesperadamente a execução. Em tais casos, a interface exibe informações disponíveis, indicando ao usuário em que ponto do intervalo/lista a extração foi interrompida (figura 8):

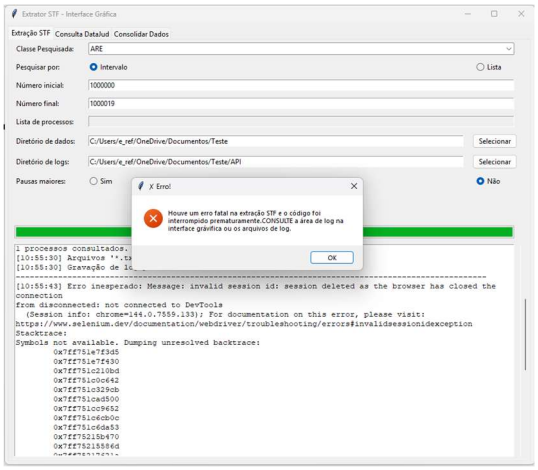

Figura 8

Também pode ocorrer fechamento automático do *WebDriver* caso haja atualização do navegador durante a execução, situação em que o painel pode não registrar detalhes completos; o usuário perceberá o erro pela ausência da janela do navegador e pela interrupção do progresso exibido na interface.

Nos testes de desenvolvimento — em que foram extraídos dados de mais de 150.000 ARE/RE —, foram identificadas essas hipóteses. Ressalta-se que os arquivos gerados (extração STF, enriquecimento *DataJud* e consolidação) foram carregados em nuvem, com *link* fornecido pelo desenvolvedor à comunidade acadêmica no final do texto deste trabalho.

Outros erros podem ser identificados futuramente, pois não foi disponibilizada versão beta, que, em geral, permite testes em escala real, identificação de bugs, avaliação de desempenho e coleta de feedback; contudo, depende de mecanismos de coleta e monitoramento de uso, economicamente inviáveis para solução não comercial e sem equipe dedicada. Ainda assim, os testes evidenciaram que o *software* cumpre adequadamente sua finalidade e tende a permanecer estável por longos períodos de execução contínua, reduzindo necessidade de monitoramento constante e liberando o pesquisador para outras tarefas, como leitura de artigos, dissertações e teses entro outras atividades de pesquisa.

5. ALGORITMOS DE ANÁLISE DE DADOS COM RETORNO DE INFORMAÇÕES CATEGORIZADAS OU DE NOVAS INFORMAÇÕES A PARTIR DE OPERAÇÕES LÓGICAS REALIZADAS SOBRE A LISTA DE ANDAMENTOS

Desde o início do desenvolvimento, pretendeu-se implementar funcionalidades que, a partir dos andamentos processuais, permitissem: (i) calcular tempo de tramitação conforme o fluxo de cada caso; (ii) identificar a decisão inicial (monocrática ou colegiada); (iii) contar e apontar recursos interpostos; (iv) verificar se alguma decisão foi reformada durante no julgamento de recursos internos; e (v) detectar julgamentos virtuais, com identificação de pedido de vista ou de destaque e do(a) Ministro(a) responsável.

A identificação e contagem de recursos internos e documentos, em termos gerais, é tarefa relativamente simples, semelhante a outras rotinas de contagem (sujeitos processuais, quantidade de andamentos, decisões e documentos). Por essa razão, as considerações mais relevantes concentram-se nos algoritmos que produzem inferências mais complexas.

5.1 Panorama dos desafios de desenvolvimento desse tipo de algoritmo

Os algoritmos descritos são utilizados pelo módulo de consolidação para tabulação e geração da planilha XLSX. Na primeira arquitetura, ainda monolítica, o desenvolvimento ainda perseguia a universalidade da aplicação para das classes processuais do STF, com a pretensão de que o programa identificasse o resultado do julgamento do recurso ou ação, além de apontar se alguma decisão proferida no curso do processo teria sido reformada ou reconsiderada. **Para efeito de modelagem**, **definiu-se** como ***"reforma"*** **o provimento** *do agravo interno* **ou** *dos embargos de divergência* **e o acolhimento** *dos embargos de declaração* (*mesmo que sem efeito infringente*, pois a decisão "reformada" foi, em alguma, medida alterada, sanando a omissão, a obscuridade ou a contradição inicialmente existente).

A lógica criada para executar essa tarefa foi a seguinte: definiu-se uma lista de eventos-chave representativos (provido, não provido, procedente, improcedente, não conhecido, determinada a devolução pela repercussão geral *etc*); percorria-se a lista de andamentos processuais e criava-se uma nova lista com os eventos identificados e o mais recente deles era tido como o resultado do julgamento; a presença de eventos como *"agravo regimental provido"*, *"embargos recebidos"*, *"reconsidero"* e *"reconsideração"* era tida como evidência de que houve reforma do ato decisório.

Conforme indicado na documentação técnica cujo *link* de acesso é indicado ao final deste relatório, foram usadas 14 (quatorze) variações indicativas de alteração do conteúdo da decisão impugnada no recurso interno (figura 9):

```
indicadores_reforma = [
    'Agravo regimental provido #',
    'Agravo regimental provido em parte #',
    'Embargos recebidos #',
    'Embargos recebidos em parte #',
    'Embargos recebidos como agravo regimental desde logo provido #',
    'Embargos recebidos como agravo regimental desde logo provido em parte #',
    'Prejudicado #',
    'Procedente #',
    'Procedente em parte #',
    'Provido #',
    'Provido em parte #',
    'Reconsidero e determino a distribuição #',
    'Reconsidero e devolvo pelo regime da repercussão geral #',
    'Reconsideração #'
]
```

Figura 9

Quanto a este ponto específico, a utilização dos marcadores os separadores "-", "#", "*", "$" e "->" para separar cada elemento do andamento ({**data**} **-** {**título**} **#** {**detalhamento**} ***** {órgão_**julgador**} **$** {**descrição_documento**} **->** {**documento**}) revelou sua verdadeira importância. A inclusão da cerquilha (#) após o título do andamento evitou que a busca utilizando o critério "Embargos recebidos #" como *substring* indicativa de acolhimento dos embargos de declaração retornasse qualquer resultado que contivesse a expressão "Embargos recebidos" acrescidas de outros termos precedendo a cerquilha, como "embargos recebidos como agravo interno".

O exame do resultado retornado pelo programa de uma amostra de mais de 46 mil processos mostrou inconsistências sérias. O problema é ilustrado pelos casos de dois recursos.

O primeiro é o ARE1349983: foi negado provimento ao recurso; houve oposição de embargos de declaração, que foram acolhidos “apenas para prestar esclarecimentos e acrescentar fundamentos à decisão recorrida, sem atribuir-lhes efeitos infringentes”. Porém a planilha devolvida pelo programa indicava que: o ARE foi provido e não houve reconsideração ou reforma da decisão, embora o detalhamento da decisão contradissesse as informações (figura 10)

Processo	(...)	Resultado do exame do recurso	Detalhamento	Reconsideração	Detalhamento reconsideração	Reforma da decisão	Detalhamento refomra
ARE 1349983	(...)	Provido	...Assim, acolho os embargos de declaração apenas para prestar esclarecimentos e acrescentar fundamentos à decisão recorrida, sem atribuir-lhes efeitos infringentes.	Não houve reconsideração.		Não houve reforma.	

Figura 10

A análise do caso e da lógica de programação aplicada revelou que a inconsistência foi causada pelo lançamento de *“provido”* (termo que, em princípio, deveria ser utilizado apenas para julgamento de mérito do apelo extremo, seja por juízo monocrático ou seja por colegiado) no andamento, em vez de *“embargos recebidos”*.

O segundo é o RE 1367071: o recurso foi provido; o agravo regimental e os embargos de divergência não foram providos. Porém, a planilha indicava que houve reconsideração, embora o seu detalhamento indicasse que a reconsideração se referia ao juízo negativo de admissibilidade do recurso de embargos de divergência, sem modificar o sentido do julgamento inicial do recurso extraordinário com agravo.

No universo desse *dataset* de teste, no qual verificou-se a interposição em interposição de recursos internos em 6.343 processos, houve lançamento inadequado do julgamento de 423 recursos internos, o que compromete significativamente a

precisão dos resultados obtidos pelo código, resultando em taxa de 6,6% de inconsistência.

De outro lado, a lógica do código, concebida com a pretensão de ter aplicabilidade universal, ignorava o fato de que, além do(a) Ministro(a) Relator(a), a Presidência também pratica atos decisórios nas classes ARE e RE. A prática demonstrou ser inviável — dentro do tempo disponível para desenvolvimento — perseguir universalidade semelhante à do módulo de raspagem, pois os fluxos de tramitação variam significativamente entre classes processuais. Essa inviabilidade decorre de diferenças de marcos de encerramento (arquivamento, baixa, devolução, remessa *etc*.), terminologias de resultado (não conhecido, provido, improvido, rejeitado, procedente, improcedente *etc*.) e presença de eventos que não ocorrem em todas as classes.

De modo simplificado, o fluxo típico de ARE/RE é: o processo é registrado à Presidência, que pode inadmitir ou determinar distribuição ao(à) Relator(a); o(a) Relator(a) pode inadmitir, julgar monocraticamente ou submeter ao colegiado; e decisões podem ser impugnadas por recursos internos. Em geral, essas classes não envolvem tutela de urgência com frequência e, quando há pedidos desse jaez, raramente são apreciados de forma relevante na cadeia decisória.

Embora os recursos extraordinários em matéria eleitoral costumam, na maioria das vezes, veicularem pedidos de tutela de urgência, esses recursos representam menos de 0,01% (zero virgula zero um por cento) ou um décimo da milésima parte dos recursos, razão pela qual, o exame desses pedidos de tutela de urgência não foram considerados no fluxo de coleta de dados de modo separado, com tabulação específica.

Em contrapartida, em ações do controle concentrado, é comum haver pedidos liminares e submissão a referendo do colegiado, com dinâmica distinta e marcos processuais próprios. Assim, a elaboração de algoritmos universais demandaria esforço desproporcional e profundo conhecimento das nuances de tramitação de cada classe; do contrário, os resultados não atingiriam confiabilidade suficiente para fundamentar pesquisa acadêmica séria.

Por essas razões, decidiu-se desenvolver a lógica dos algoritmos mais complexos tendo como referência o fluxo específico de ARE/RE, que, além de constituir objeto do projeto desde a origem, corresponde à experiência profissional acumulada pelo desenvolvedor ao longo de quase uma década de trabalho com essas classes.

Ressalte-se, contudo, que a existência de módulo de extração com uso potencialmente amplo abre caminho para evolução futura: o módulo de consolidação pode, em versões posteriores, chamar funções específicas de cálculo/análise de acordo com a classe processual extraída, agregando lógicas distintas a serem desenvolvidas para cada classe.

Mesmo com essas limitações, a aplicação mantém utilidade para extração e consolidação tabular de outras classes, desde que o pesquisador desconsidere colunas cujos valores dependem de algoritmos calibrados exclusivamente para o fluxo ARE/RE. Assim, na versão atual, recomenda-se desconsiderar, para outras classes processuais, os campos relacionados ao status e duração de tramitação e aqueles vinculados a atos sob Presidência/Relatoria e a indicadores de reforma, conforme indicado neste relatório.

5.2 Algoritmo cálculo do tempo de tramitação dos recursos extraordinários e dos recursos extraordinários com agravo

Para cada recurso extraordinário ou recurso extraordinário com agravo, o código analista a cadeia de andamentos extraídas por meio do processo de raspagem de dados já descrito, identifica se a tramitação do processo se encerrou, ou não, classificando o tipo de desfecho em doze categorias específicas:

i. autuação cancelada;
ii. autuação retificada;
iii. baixa com trânsito em julgado;
iv. baixa definitiva - motivo diverso;
v. devolução em razão de representativo da controvérsia;
vi. devolução pela repercussão geral;
vii. devolução por impossibilidade de processamento;

viii. devolução por remessa indevida;
ix. em tramitação;
x. processo findo - motivo diverso;
xi. reautuado; e
xii. remessa externa - motivo diverso.

O tempo de tramitação é calculado pela diferença, em dias, entre a data de protocolo e a data do evento que marcou o fim do processo, excetuados os casos ainda em tramitação, em que não se retorna valor final. A lógica foi moldada para o fluxo típico de ARE/RE no STF e depende de marcadores textuais: de recebimento, baixa e remessa; do andamento “processo findo” como indicativo de cancelamento, retificação ou reautuação; e de marcadores de devolução ou trânsito em julgado.

Esse desenho permite ao pesquisador calcular tempos médios com filtros, excluindo processos em que houve reautuação ou cancelamento/retificação da autuação, ou segmentando por tipo de encerramento (por exemplo, devolução para aguardar paradigma de repercussão geral ou trânsito em julgado), com a possibilidade de, ainda separar, os recursos com julgamento de mérito daqueles inadmitidos por óbices processuais.

5.3 Algoritmo de separação dos atos praticados sob as direções da presidência e do relator

Os resultados desse algoritmo são essenciais para os módulos que examinam decisões e verificam reforma, pois tanto a Presidência quanto o(a) Relator(a) podem praticar atos decisórios relevantes. A primeira tentativa de implementação — baseada em busca simples por termos nos andamentos — revelou imprecisões graves, levando ao reprojeto da lógica e, indiretamente, à evolução arquitetural do sistema.

Na versão inicial monolítica, o algoritmo buscava títulos/expressões em listas pré-definidas para inferir decisão inicial, recursos interpostos, reconsideração e eventual reforma, retornando também observações do andamento quando presentes. No *dataset* de teste inicial, os resultados exibiram falhas relevantes, com elevada taxa de imprecisão, como já ressaltando.

Exemplo emblemático foi o ARE 1349983: embora embargos de declaração tenham sido acolhidos *"apenas para esclarecimentos"* (sem efeitos infringentes), a classificação retornada na planilha sugeria inexistência de reforma, pois o andamento foi lançado como *"provido"* em vez de *"embargos recebidos"*, contrariando a metodologia estabelecida para a modelagem, segundo a qual se definiu como *"reforma"* o provimento do agravo interno ou dos embargos de divergência e o acolhimento dos embargos de declaração (mesmo que sem efeito infringente)

Diante disso, a lógica foi reestruturada: o novo algoritmo opera como *"classificador de linha do tempo"*, filtrando, dentre todos os andamentos, apenas os eventos juridicamente relevantes (decisões e interposição de recursos internos) e formando nova lista. Em seguida, *"corta"* a linha do tempo no marco em que o processo sai da Presidência (geralmente indicado por *"Distribuição"*), separando eventos anteriores (sob Presidência) e posteriores (sob relatoria).

A atuação da Presidência é indicada pelo marcador *"registrado à Presidência"*; a atuação do(a) Relator(a) é indicada pela presença de *"distribuído"* em algum andamento (inclusive variações como *"distribuído por prevenção"*). Na ausência de tais marcadores, o sistema registra "não registrado à Presidência" ou "não distribuído", conforme o caso. As listas resultantes são então passadas como argumentos obrigatórios às funções de análise de decisões e de verificação de reconsideração/reforma.

5.4 Algoritmos de análise das decisões proferidas pelo(a) Presidente e pelo Ministro(a) Relator(a)

Como o lançamento de andamentos no STF nem sempre observa rigor terminológico, não se mostrou confiável fundamentar a lógica apenas em *strings* específicas como *"reconsideração", "embargos recebidos", "agravo regimental provido" etc*. Uma solução possível seria empregar LLM para classificar o "sentido" dos títulos, replicando a metodologia usada por Guilherme Ramos de Morais e pelo Dr. Henrique Araújo Costa na produção do ensaio *"Direito e Inteligência Artificial: Análise de Sentimento Aplicada em Certidões de Julgamento de Mandados de Segurança Impetrados no Supremo Tribunal Federal"* apresentado no Encontro

Nacional de Administração da Justiça[9]. Contudo, isso contrariaria o objetivo central do projeto, que é disponibilizar ferramenta gratuita à comunidade, sem impor custos de processamento por *APIs* de modelos proprietários. Ademais, permitir múltiplos LLMs implicaria aumentar complexidade (*APIs* distintas, *endpoints* distintos, parâmetros distintos), contrariando a filosofia de simplicidade e legibilidade (*"Zen do Python"*).

A análise de grandes volumes (mais de 100.000 processos) sugeriu que os títulos dos andamentos são "rígidos", pois selecionados de lista predefinida pelo sistema do STF; isso reduz variação linguística e permite abordagem heurística por padrões sem depender de IA generativa. O desenvolvedor obteve, inclusive, planilha interna com 330 títulos possíveis, o que reforça a finitude do vocabulário de lançamentos.

Isso significa que o servidor responsável por o fazer não pode inventar um título para o andamento processual a ser lançado, fator, em alguma medida, padroniza os andamentos, pois só pode utilizar *"provido em parte"* ou *"não provido"*, em vez de, por exemplo, *"parcialmente provido"*, *"desprovido em parte"* ou *"improvido"* e suas variações.

A limitação da quantidade de termos ou expressões para indicar a prática de um ato processual no feito inviabiliza a criação de variações para expressar a mesma ideia, ou seja, há uma finitude de termos expressões que podem ser utilizados no lançamento dos andamentos processuais.

Com inspiração na técnica de análise de sentimento em NLP (Processamento de Linguagem Natural), usada para identificar e classificar automaticamente as emoções ou opiniões expressas em um texto, a estratégia adotada foi atribuir polaridade (favorável/desfavorável) a rótulos de resultado, em relação à ocorrência de reforma. Definiu-se lista de marcadores "positivos" (p. ex*., "provido", "procedente", "reconsideração" etc.*), e as funções que analisam decisões percorrem eventos relevantes após a decisão inicial, verificando se algum marcador de reforma aparece

[9] MORAIS, G. R.; COSTA, H. A.. Direito e Inteligência Artificial: Análise de Sentimento Aplicada em Certidões de Julgamento de Mandados de Segurança Impetrados no Supremo Tribunal Federal, 2020, Curitiba. Anais do EnAJUS (Encontro Nacional de Administração da Justiça). Disponível em: https://www.enajus.org.br/anais/assets/papers/2020/sessao-14/2-direito-e-intelige-ncia-artificial-ana-lise-de-sentimento-aplicada-em-certido-es-de-julgamento-de-mandados-de-seguranc-a-impetrados-no-supremo-tribunal-federal.pdf. Acesso em: 9 fev. 2026.

nos eventos subsequentes. Quando identificado, registra-se ocorrência de reforma, quantifica-se o número de reformas e captura-se, quando possível, o detalhamento e o órgão prolator do ato reformador, explorando a estrutura padronizada do andamento (campos separados por marcadores).

Assim, a função recebe como argumento obrigatório a lista de eventos relevantes gerada por função específica e toma o primeiro item como a decisão inicial do processo proferida pela Presidência ou pelo(a) Ministro(a) Relator(a), conforme o caso; em seguida, itera sobre os demais à procura de marcadores de reforma.

Todavia, a simples busca usando a função "*.find()"* pode gerar falsos positivos. Exemplo: no RE 1367071, a reconsideração ocorreu na admissibilidade de embargos de divergência (recurso interno), mas a reforma relevante para a pesquisa é a que incide sobre a decisão que resolveu o recurso extraordinário (inadmissão ou exame do mérito), e não sobre admissibilidade de recursos internos. Para evitar falsos positivos, a solução mais adequada foi adotar *Regex* (expressões regulares), permitindo declarar padrões que exigem a presença do marcador de reforma e, simultaneamente, negam expressões indicativas de juízo admissibilidade de embargos de divergência no mesmo andamento (como "não admito" / "não admitir" / "inadmito" / "inadmitir" e "embargos de divergência" / "embargos de divergência", e outras possíveis variações).

Com essa lógica, a precisão dos resultados aumentou substancialmente, elevando a confiabilidade dos dados. O exame dos resultados obtidos a partir da raspagem de ados de cerca de 150 mil processos revelou que o novo algoritmo eliminou 100 % distorções decorrentes da falta de padronização no lançamento dos andamentos. Não se pode afirmar, todavia, que a precisão do resultado é de 100%, pois há casos em que a coluna "BB" da planilha não contém o detalhamento da reforma, isso porque em alguns processos o próprio gabinete deixou de lançar o dispositivo da decisão no andamento. Desse modo, a conferência da precisão depende da consulta ao documento da decisão reformadora.

É o caso, por exemplo, do RE 1350683, no qual o Ministro Nunes Marques acolheu os embargos de declaração e foi lançado o andamento datado de 18/4/2022 o andamento "Embargos recebidos", porém não se detalhou o andamento. Neste caso, não há a possibilidade de analisar, apenas pelos dados dos andamentos, a

correção do lançamento “Embargos recebidos”. É preciso que se consulte o documento da decisão do recurso para se aferir a correção do lançamento na página de consulta processual, o que não se fez para cada um dos processos cujos dados foram raspados.

Em termos conceituais, as funções “analise_decisoes_presidencia()” e “analise_decisoes_relator()” operam como heurísticas textuais replicáveis que analisam os apenas os eventos após a primeira decisão sob a direção da Presidência e do(a) Ministro(a) Relator(a) – conforme a linha do tempo de eventos relevantes (cujo recorte é gerado pela função “eventos_presidência_e_relator()” – e respondem: (i) se houve reforma da primeira decisão sob Presidência e/ou relatoria; (ii) quantas reformas ocorreram; e (iii) quais eventos correspondem às reformas, com indicação do órgão.

O algoritmo, contudo, não mede a intensidade da reforma: há casos em que provimento de agravo altera apenas honorários recursais, sem modificar o mérito ou acolhimento de embargos se dá tão somente para esclarecer o julgado, sem modificá-lo. Por isso, se o pesquisador precisar qualificar a *“medida”* da reforma, será necessário examinar diretamente o conteúdo da decisão reformadora. Na maior parte dos processos, os gabinetes lançam a parte dispositiva da decisão no andamento, mas há casos em que isso não é feito, o que demandará a abertura manual dos arquivos .PDF dos documentos e a sua leitura.

6. DESENVOLVIMENTO DO CÓDIGO COM AUXÍLIO DE MODELOS DE IA GENERATIVA E IMPRESSÕES COLHIDAS DURANTE O PERCURSO

Para escrever o código, foi necessário que o desenvolvedor aprendesse a linguagem *Python*. Como fontes de aprendizado, utilizaram-se cursos gratuitos disponíveis em canais do *YouTube*, especialmente Hashtag Treinamentos[10] e Bóson Treinamentos[11]. Esses materiais forneceram compreensão suficiente dos conceitos fundamentais da linguagem e do uso de suas bibliotecas.

Ferramentas de inteligência artificial, como ad plataforma *Perplexity* (plano Pro Básico) e *Copilot*, mostraram-se úteis para a correção de sintaxe e para ajustes pontuais na lógica de determinados trechos. Contudo, não se revelaram tão eficazes na elaboração completa da lógica de programação de algumas funções.

Ilustrativamente, em determinado momento, buscava-se armazenar números de processos que não correspondessem nem a ARE nem a RE. O código inicialmente gerado pela IA produzia resultados inconsistentes, registrando processos válidos como inexistentes.

O prompt enviado à IA descrevia o fluxo esperado: caso um número não correspondesse a um processo da classe ARE, a variável *"classe"* deveria ser alternada para *"RE"* e a consulta repetida. Somente se não houvesse correspondência em nenhuma das duas classes, o número deveria ser incluído na lista de processos inexistentes. Entretanto, os códigos sugeridos continham mais de 20 linhas de sintaxe pouco compreensível, o que inviabilizava sua incorporação ao programa principal.

Após reflexão adicional, o desenvolvedor elaborou uma solução simples, composta por apenas quatro linhas de código, plenamente compreensíveis, pessoas sem conhecimento avançado em *Python*:

```
lista_provisória = []
(...)
lista_provisória_are.append(processo) # Acrescenta o apenas número do processo consultado na variável auxiliar do tipo lista.
```

[10] URL: https://www.youtube.com/@HashtagTreinamentos
[11] URL: https://www.youtube.com/@bosontreinamentos

```
            if lista_provisória_are.count(processo) == 2: # Verifica se o número do processo aparece duas vezes na variável auxiliar do tipo lista.
                        classe_e_numeros_inexistentes.append(f'ARE ou RE {processo}') # Se sim, acrescenta a "ARE ou RE" e o número do processo na variável "classe_e_numero_inexistentes"
            lista_provisória_are =[]
```

O funcionamento do código acima é o seguinte: antes do início do *loop*, cria-se uma lista auxiliar vazia. Durante a consulta, se o processo não for encontrado na classe ARE, o número é armazenado nessa lista. Caso a consulta à classe RE também não retorne resultado, o mesmo número é novamente incluído, totalizando duas ocorrências. Nesse momento, o bloco condicional é ativado, e o número é registrado na lista de processos inexistentes com o prefixo “ARE ou RE”. Em seguida, a lista auxiliar é esvaziada, evitando consumo desnecessário de memória.

A solução criada a partir da lógica do desenvolvedor é simples e, portanto, *“melhor do que complexa”,* resultando em um código mais *“pythônico”* do que aquele gerado pelas ferramentas de inteligência artificial, em conformidade com um dos pilares fundamentais da filosofia Python de programação. Contundo, não se pode deixar de destacar que, quando os *prompts* foram cuidadosamente elaborados, com descrição detalhada do que se pretendia alcançar e da forma como os resultados divergiam do objetivo, as ferramentas de IA mostraram-se bem eficientes na correção da lógica ou da sintaxe, revelando-se complementares ao processo de desenvolvimento.

7. USOS POTENCIAIS E CENÁRIOS DE APLICAÇÃO

A entrega, pelo programa, em formato tabulado de 61 (sessenta e um) parâmetros extraídos de cada ARE/RE (incluindo informações como informações como número do processo, relator, partes envolvidas, andamentos, decisões, recursos internos, eventos relevantes e detalhes sobre julgamentos virtuais), oferecem base de dados sólida para estudos de jurimetria e pesquisa empírica sobre tempo de tramitação, recorribilidade interna, reformas de decisões, taxa de sucesso dos recursos internos, padrões de andamentos, atuação da Presidência e dos Relatores (volume de decisões, tipos de atos, índice de reforma).

O enriquecimento dos dados do STF com dados da *API DataJud* permite a identificação exata do órgão de origem da controvérsia objeto dos recursos, o tipo de ação/recurso de cujo julgamento se recorreu extraordinariamente e o tema objeto da controvérsia na origem, em complementação ao assunto do ARE/RE.

Os dados permitem a identificação de (i) processos julgados em ambiente virtual (início/fim/suspensão), (ii) quem pediu destaque em julgamentos virtuais e em quais casos; e (iii) quem pediu vista e em quais processos, com potencial correlação com atrasos na tramitação; informações que podem orientar estudos comparativos sobre produtividade dos Ministros e dos órgãos colegiados, abrangendo sessões de julgamento realizadas nos ambientes virtuais e físicos.

Seria possível também verificar também, por exemplo, quais os tribunais de origem dos recursos devolvidos pela sistemática da repercussão geral, em ordem a possibilitar uma investigação dos motivos que levam estes órgãos judiciários a remeter ao Supremo Tribunal Federal recursos que tem como objeto controvérsias já afetadas à sistemática da repercussão geral.

Poder-se-ia, além disso, identificar quais são os tribunais cujas decisões sofrem mais reforma, revelando uma possível resistência na aplicação da jurisprudência do Supremo Tribunal Federal e quais são as matérias debatidas nestes processos.

No mesmo sentido, seria possível proceder a estudos sobre quem são as partes que mais recorrem à Suprema Corte ou quais são os advogados que obtém as maiores taxas de sucesso em seus recursos.

Todos os 61 parâmetros tabulados permitem combinar diversos dados para produção de inúmeros estudos.

O formato JSON, acrescido da estruturação dos dados com marcadores definidos, contendo, inclusive, *links* das decisões e de outros documentos relevantes disponibilizados na página de andamentos dos processos facilita a formação de *dataset* para treinamento de sistemas baseados em rede neural, com o objetivo de identificar padrões decisórios.

Embora a fonte dos dados consolidados pelo sistema seja publica (portal do STF), deve-se observar à disciplina imposta pela Lei Geral de Proteção de Dados – LGPD, que não se aplica ao tratamento de dados pessoais para fins exclusivamente particulares e não econômicos. Para fins acadêmicos, os dados devem ser anonimizados ou descartados quando não relevantes para a análise acadêmica.

8. CONCLUSÃO

Este sistema constitui uma solução robusta e bem estruturada para a extração e consolidação de dados processuais, revelando-se particularmente valioso para pesquisadores, profissionais do direito e pós-graduandos da área. Sua lógica de extração e interpretação de dados do STF é sofisticada, resolvendo problemas práticos como instabilidade de conexão e complexidade na leitura de andamentos processuais. A arquitetura modular, embora apresente algumas limitações, assegura flexibilidade e facilidade de manutenção. As estratégias de tratamento de exceções e de recuperação de falhas encontram-se relativamente bem desenvolvidas, conferindo estabilidade ao programa — aspecto relevante diante da natureza volátil das operações de raspagem de dados na internet — e permitindo a extração escalável de informações sem necessidade de supervisão humana constante.

Os principais diferenciais do sistema residem na integração entre dados provenientes do STF e dos tribunais de origem, o que enriquece o conjunto de dados formado; na lógica de programação que possibilita uma visão abrangente do ciclo de vida processual dos recursos extraordinários e dos recursos extraordinários com agravo; e na disponibilização de uma distribuição com interface gráfica integrada, gerada a partir do empacotamento do código-fonte em Python, que permite a execução do *software* independentemente da instalação de interpretador, garantindo que usuários sem conhecimento prévio em programação possam utilizá-lo.

Os módulos responsáveis pela raspagem de dados via *WebDriver* e pela extração de dados *via API DataJud* estão adaptados para coleta de dados de qualquer das classes processuais do Supremo Tribunal Federal, formando um *dataset* organizado e estruturado, com delimitadores de texto padronizados que facilitam a análise e a consolidação dos dados pelos demais módulos.

Embora a lógica de consolidação dos dados tenha sido desenvolvida especificamente para capturar as nuances do fluxo de tramitação das classes processuais ARE e RE, a arquitetura modular do código permite seu aprimoramento e expansão, de modo a incorporar lógicas voltadas ao tratamento do fluxo processual

de outras classes, assegurando adaptabilidade e potencial de generalização do sistema ao longo do tempo.

9. ARQUIVOS ARMAZENADOS NA NUVEM

Foram disponibilizados os seguintes arquivos:

1. **Arquivos JSON** contendo dados extraídos dos AREs e REs recebidos pelo Supremo Tribunal Federal (STF) durante o último ano da Presidência do Ministro Luiz Fux;
2. **Arquivos JSON** contendo dados extraídos dos AREs e REs recebidos pelo STF durante o primeiro ano da Presidência da Ministra Rosa Weber;
3. **Arquivos XLSX e JSON** de consolidação dos dados referentes aos dois períodos mencionados;
4. **Distribuição autoexecutável** do programa desenvolvido; e
5. **Documentação técnica.**

9.1 Links de acesso

Todos os arquivos podem ser baixados por meio do seguinte *link* de acesso:

https://1drv.ms/f/c/c5e1823c8be8ded8/IgCgDngBUX5DR7orEqMivNoOAbjmz7TUE-PXgeyKt-GCwpE?e=NQbYjO

9.2 Repositório do código-fonte

Os códigos-fonte em Python, correspondentes a cada um dos módulos implementados, encontram-se disponíveis no repositório GitHub do autor:

https://github.com/eloireffatti/STF-Selenium-com-GUI

REFERÊNCIAS

STF. **STF faz chamamento público para projetos de inteligência artificial que automatizem resumos de processos**. Brasília, 7 de novembro de 2023. Disponível em https://noticias.stf.jus.br/postsnoticias/stf-faz-chamamento-publico-para-projetos-de-inteligencia-artificial-que-automatizem-resumos-de-processos/. Último acesso em 17 de fevereiro de 2026.

COSTA, Alexandre e BENVINDO, Juliano, A Quem Interessa o Controle Concentrado De Constitucionalidade? - O Descompasso entre Teoria e Prática na Defesa dos Direitos Fundamentais (*Who is Interested in the Centralized System of Judicial Review? - The Mismatch between Theory and Practice in the Protection of Basic Rights*) (1 de abril de2014). Disponível na SSRN: https://ssrn.com/abstract=250954 ou http://dx.doi.org/10.2139/ssrn.2509541. Último acesso em 17/02/2026.

COSTA, Alexandre Araújo. 2026. [GitHub repository]. GitHub. Disponível em: < https://github.com/AlexandreAraujoCosta?tab=repositories >. Último acesso em 17/02/2026.

OPEN AI. *Pricing*. 2026. Disponível em https://platform.openai.com/docs/pricing?latest-pricing=standard. Último acesso em 17/02/2026.

STF. 2026. Portal Corte Aberta. Disponível em https://transparencia.stf.jus.br/extensions/corte_aberta/corte_aberta.html. Último acesso em 17/02/2026.

PEIXOTO, Fabiano Hartmann; BONAT, Debora. GPTs e Direito: impactos prováveis das IAs generativas nas atividades jurídicas brasileiras. Sequência Estudos Jurídicos e Políticos, Florianópolis, v. 44, n. 93, p. 1–31, 2023. DOI: 10.5007/2177-7055.2023.e94238. Disponível em: https://periodicos.ufsc.br/index.php/sequencia/article/view/94238. Último acesso em 17/02/2026.

YALE. *AI at Yale*. 2026. Disponível em https://ai.yale.edu/yales-ai-tools-and-resources/clarity-platform/formatting-files. Último acesso em 17/02/2026.

MORAIS, G. R.; COSTA, H. A.. Direito e Inteligência Artificial: Análise de Sentimento Aplicada em Certidões de Julgamento de Mandados de Segurança Impetrados no Supremo Tribunal Federal, 2020, Curitiba. Anais do EnAJUS (Encontro Nacional de Administração da Justiça). Disponível em: https://www.enajus.org.br/anais/assets/papers/2020/sessao-14/2-direito-e-intelige-ncia-artificial-ana-lise-de-sentimento-aplicada-em-certido-es-de-julgamento-de-mandados-de-seguranc-a-impetrados-no-supremo-tribunal-federal.pdf. Último acesso em 17/02/2026.

HASHTAG TREINAMENTOS. Disponível em https://www.youtube.com/@HashtagTreinamentos. Último acesso em 17/02/2026.

BOSON TREINAMENTOS. Disponível em https://www.youtube.com/@bosontreinamentos. Último acesso em 17/02/2026.