Grafo de Prioridade para Pré-Processamento de Texto

Gustavo Alexandre Sousa Santos

[1]Instituto da Computação – Universidade Federal Fluminense (UFF)
Niterói – RJ – Brazil

Abstract. *This article proposes an optimized preprocessing pipeline for Large Language Models (LLMs) applied to the legal domain. The methodology employs a dependency graph with a priority queue to model and manage tasks, enabling dynamic scheduling and parallel execution, aiming to maximize computational resource utilization and preserve data flow integrity. The system, implemented in C++, was rigorously evaluated using a set of legal documents, comparing three execution strategies: sequential, scheduler-based parallelism, and data partitioning parallelism. The results demonstrate that the data partitioning strategy achieved the most significant performance, reaching a 500% performance gain and a throughput of 53,641 documents per second. It is concluded that sequential dependencies impose severe limitations on effective parallelism, and that data partitioning represents a highly effective approach to optimize the preprocessing of extensive textual volumes, contributing to more efficient LLM training within the legal context.*

Resumo. *Este artigo propõe um pipeline de pré-processamento para Modelos de Linguagem de Grande Escala (LLMs) aplicados ao setor jurídico, utilizando um grafo de dependência com fila de prioridade para modelar tarefas e suas dependências. A abordagem permite o escalonamento dinâmico e a execução paralela de tarefas, otimizando o uso de recursos computacionais e garantindo a integridade do fluxo de dados. O sistema, implementado em C++, foi avaliado com um conjunto de documentos jurídicos. Três estratégias de execução foram comparadas: sequencial, paralelismo baseado em scheduler e paralelismo com particionamento de dados. O particionamento de dados apresentou o melhor desempenho, alcançando um desempenho de 500% e throughput de 53.641 documentos por segundo. Os resultados evidenciam que dependências sequenciais limitam drasticamente o paralelismo efetivo e que o particionamento de dados é uma estratégia eficaz para otimizar o pré-processamento de grandes volumes de texto, contribuindo para o treinamento de LLMs de forma mais eficiente no contexto jurídico.*

1. Introdução

Os Modelos de Linguagem de Grande Escala (do inglês, *Large Language Models*) como, por exemplo, GPT, Gemini, Claude, Grok, DeepSeek, etc., representam um marco significativo no desenvolvimento da Inteligência Artificial, demonstrando aplicabilidade em múltiplos domínios, desde processamento de linguagem natural até geração de conteúdo especializado. A adaptação desses modelos para domínios específicos, particularmente o setor jurídico, apresenta desafios metodológicos complexos relacionados à qualidade e

estruturação dos dados de entrada. A eficácia de LLMs em contextos legais está intrinsecamente vinculada a processos rigorosos de pré-processamento que assegurem conformidade com as especificidades terminológicas, sintáticas e semânticas do discurso jurídico.

O processo de ajuste (*fine-tuning*) desses modelos LLM para aplicações jurídicas demanda curadoria textual sistemática e especializada, incluindo desde legislação até sentenças judiciais. A ausência de preparação adequada dos dados — envolvendo remoção de elementos não textuais, normalização linguística e segmentação semântica — compromete significativamente o desempenho dos modelos resultantes. O volume substancial de dados jurídicos, frequentemente na ordem de terabytes, estabelece o pré-processamento como um gargalo computacional crítico no desenvolvimento de LLMs especializados.

A orquestração eficiente de tarefas de pré-processamento constitui o problema central desta investigação. Arquiteturas de pipeline sequenciais demonstram limitações de escalabilidade em termos de tempo de execução e utilização de recursos computacionais (Lee et al. 2021). Simultaneamente, a interdependência inerente entre etapas de processamento exige controle rigoroso da ordem de execução, enquanto a existência de tarefas independentes oferece oportunidades para paralelização (Verhaeghe et al. 2024). A carência de metodologias sistemáticas para gerenciamento de dependências e exploração de paralelismo resulta em pipelines subótimos que impactam negativamente a eficiência operacional e os custos computacionais.

Este trabalho propõe um fluxo de pré-processamento de dados jurídicos baseado em Grafo de Dependência com Fila de Prioridade como solução para os desafios identificados. A metodologia fundamenta-se na modelagem de tarefas de pré-processamento como um grafo acíclico dirigido, onde vértices representam operações de processamento e arestas codificam relações de precedência e fluxo de dados. Dessa forma, a proposta visa alcançar os seguintes objetivos:

- Desenvolver uma arquitetura computacionalmente eficiente para pré-processamento de dados em contextos de *fine-tuning* de LLMs;
- Demonstrar a aplicabilidade de grafos de dependência na modelagem de *workflows* de processamento com execução paralela;
- Implementar algoritmos de escalonamento baseados em filas de prioridade para otimização dinâmica de recursos;
- Avaliar a performance da solução proposta no processamento de corpora jurídicos de grande escala.

Além desta introdução, este artigo está estruturado da seguinte forma: a Seção 2 apresenta os trabalhos relacionados. A Seção 3 apresenta a metodologia desenvolvida no pré-processamento de texto baseado em grafo com fila de prioridade. Em seguida, a Seção 4 evidencia os resultados alcançados e o comparativo de desempenho entre os cenários avaliados. Por fim, a Seção 5 destaca as conclusões e considerações finais sobre este estudo.

2. Trabalhos Relacionados

Diversos trabalhos recentes abordam o escalonamento eficiente de tarefas para LLMs, *Deep Learning* e Ciência de Dados, com ênfase no uso de filas de prioridade e estruturas baseadas em grafos.

Fu et al. 2024 propõem uma abordagem inovadora para o escalonamento de requisições em LLMs utilizando técnicas de *learning to rank*. O método desenvolvido aproxima o desempenho do algoritmo *Shortest-Job-First* (SJF), tradicionalmente considerado ótimo quando os comprimentos das tarefas são conhecidos. Os resultados experimentais demonstram ganhos expressivos, com redução de até 2,8 vezes na latência em aplicações de chatbot e aumento de até 6,5 vezes no throughput em cenários de geração de dados sintéticos.

O trabalho de Wei et al. 2025 apresentou uma arquitetura baseada em filas duplas (*dual-queue*) para o escalonamento de tarefas em LLMs. O estudo visou distinguir entre requisições reativas, que exigem resposta imediata, e requisições proativas, que toleram maior latência, utilizando para isso uma decomposição de tarefas fundamentada em grafos heterogêneos de execução (*Heterogeneous Execution Graphs* - HEG).

Já Ren 2024 investigou o escalonamento de tarefas em ambientes descentralizados de LLM *serving*, modelando o problema como um grafo direcionado. Nessa modelagem, os vértices representam operações de computação em diferentes camadas do modelo e máquinas, enquanto as arestas correspondem à latência de comunicação. O escalonamento é realizado por meio de filas de prioridade ordenadas por distância, permitindo a aplicação eficiente de algoritmos de ciclo mais curto.

Por outro lado, o artigo de Mitzenmacher and Shahout 2025 discutiu os desafios e as oportunidades do uso de predições de aprendizado de máquina em sistemas de filas para LLMs. O estudo destacou questões como variabilidade nos tempos de inferência, limitações dinâmicas de memória (especialmente em estruturas *key-value*), e a necessidade de múltiplas estratégias de preempção para garantir desempenho e eficiência.

3. Metodologia

Esta seção apresenta a metodologia para desenvolvimento de um sistema de escalonamento e execução eficiente de tarefas de pré-processamento de texto para otimização do treinamento de LLMs. A abordagem baseia-se na representação das tarefas como grafo de dependências e utilização de fila de prioridades para maximizar paralelismo e *throughput*. Para tal objetivo, será utilizado o conjunto de dados disponível no repositório do projeto chamado docs.csv, contendo documentos de textos jurídicos.

3.1. Processamento de Texto para LLMs

As tarefas de pré-processamento foram projetadas para otimizar a entrada para LLMs, abordando variabilidade de formatos, ruídos e necessidade de representações consistentes, conforme o fluxo de informação apresentado na Figura 1.

- **Limpeza de Texto:** Remoção de caracteres especiais, tags HTML, URLs e múltiplos espaços para redução de ruídos e padronização.
- **Normalização:** Conversão para minúsculas, tratamento de contrações e padronização de sinônimos.
- **Tokenização:** Divisão em tokens usando *Byte Pair Encoding* (BPE) ou WordPiece para balancear vocabulário e capacidade de lidar com palavras raras.
- **Preparação para *Embedding*:** Sequenciamento, truncamento ou *padding* dos tokens com adição de tokens especiais (`[CLS]`, `[SEP]`, `[EOF]`).

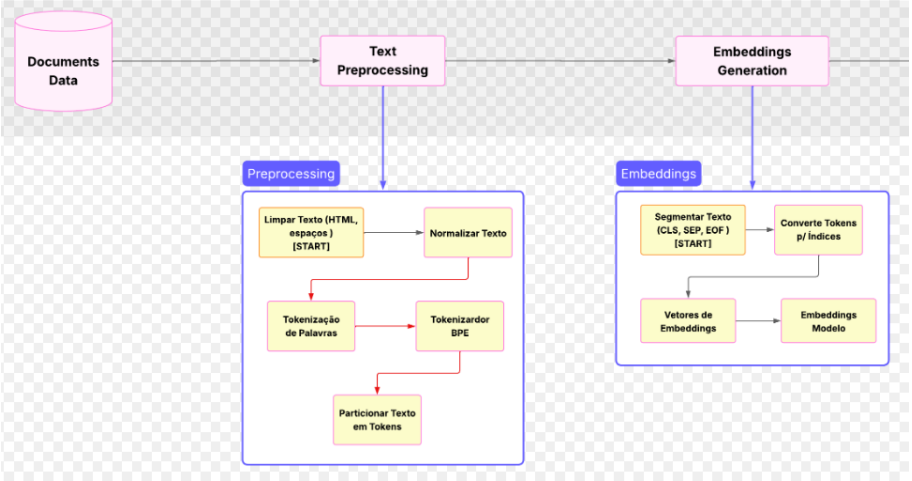

Figura 1. Visão geral das Etapas do Pré-Processamento.

3.2. Arquitetura do Sistema

A abordagem adota um projeto modularizado com cinco componentes principais, a fim de representar as etapas de pré-processamento destacas na Figura 1.

Para atender às etapas do pré-processamento identificadas anteriormente, foi implementado um grafo de dependências baseado em uma fila de prioridades, conforme as seguintes funcionalidades:

1. **Módulo de Leitura e *Parsing*:** Responsável pela ingestão de dados textuais brutos (e.g., `docs.csv`) e preparação para etapas subsequentes.
2. **Gerador de Grafo de Tarefas:** Constrói um *Directed Acyclic Graph* (DAG) onde nós representam tarefas de pré-processamento e arestas indicam dependências. A granularidade configurável permite decomposição em subtarefas independentes.
3. **Módulo de Priorização:** Atribui prioridades estáticas ou dinâmicas às tarefas baseadas em criticidade, tamanho dos dados e dependências. Utiliza fila de prioridades para ordenação das tarefas prontas para execução.
4. **Módulo de Execução Paralela:** Consome tarefas da fila e as despacha para execução paralela via múltiplos *threads*, com mecanismos de sincronização para integridade dos dados.
5. **Gerenciador de Pipeline:** Orquestra o fluxo de dados e coordena a execução entre módulos, garantindo ordem correta e transferência adequada de resultados intermediários.

A metodologia utiliza representação em grafo para modelar dependências entre tarefas. O DAG identifica tarefas executáveis em paralelo e aquelas que requerem conclusão de predecessores. A gestão emprega fila de prioridades onde tarefas com dependências satisfeitas são inseridas após a conclusão de predecessores. A seleção é baseada em prioridade, priorizando tarefas críticas. A alocação para *workers* paralelos segue estratégia gulosa, selecionando tarefas de maior prioridade disponível.

4. Resultados

Os experimentos realizados com o pipeline de pré-processamento de documentos jurídicos demonstraram diferenças significativas entre as três estratégias de

implementação avaliadas. O dataset utilizado consistiu em 47.972 documentos jurídicos, processados em um ambiente controlado com 8 núcleos de processamento disponíveis.

A Tabela 1 apresenta os resultados quantitativos obtidos para cada estratégia implementada, incluindo tempo de execução, *speedup* e *throughput*.

Tabela 1. Comparação de Performance entre Estratégias de Paralelização

Estratégia	Tempo (s)	Speedup	Throughput (docs/s)
Sequencial	4,5612	1,00x	10.517
Scheduler (Paralelo)	4,7876	0,95x	10.020
Particionamento	0,8943	5,10x	53.641

O speedup obtido pela estratégia de particionamento de dados alcançou **5,10x** em relação à execução sequencial, aproximando-se do speedup linear teórico de 8x para o *hardware* de 8 processadores utilizado. A eficiência de 63,8% indica uma utilização satisfatória dos recursos computacionais disponíveis, considerando os *overheads* inerentes à paralelização.

Em contraste, a estratégia baseada em scheduler com grafo de dependências apresentou performance inferior à execução sequencial, com desempenho de apenas 0,95x. Este resultado evidencia que o *overhead* de sincronização e gerenciamento de *threads* superou os benefícios potenciais da paralelização para este tipo específico de *workload*.

4.1. Análise Detalhada por Estratégia

A **estratégia sequencial (*baseline*)** serviu como baseline para comparação, processando os 47.972 documentos em 4,5612 segundos. Esta abordagem apresentou comportamento linear previsível, com throughput de 10.517 documentos por segundo. A execução sequencial demonstrou estabilidade e consistência, sem overhead de sincronização ou gerenciamento de threads.

A **estratégia baseada em *scheduler* paralelizado** apresentou tempo de execução de 4,7876 segundos, resultando em performance 5% inferior à execução sequencial. Os logs de execução revelam que o pipeline opera de forma essencialmente sequencial devido às dependências rígidas entre etapas:

1. CleanText → NormalizeText → WordTokenization
2. BPETokenization → PartitionTokens → AddSpecialTokens
3. TokensToIndices → GenerateEmbeddings

A **estratégia de particionamento de dados paralelizado** demonstrou performance superior, alcançando o menor tempo de execução (0,8943 segundos) e maior *throughput* (53.641 documentos/segundo). O algoritmo dividiu o dataset em 48 chunks de aproximadamente 1.000 documentos cada, processados independentemente por 8 *threads*.

Os fatores de sucesso identificados no presente estudo demonstraram a efetividade da abordagem na otimização do desempenho, no qual o paralelismo através do particionamento dos dados, alcançado através da independência dos chunks, mitigou as dependências entre *threads* de processamento, contribuindo significativamente para a escalabilidade da execução. Com isso, evidenciou-se que a distribuição uniforme do *workload*

promoveu um balanceamento eficiente de carga, o que somado a operação de cada *thread* em segmentos contíguos de memória otimizou a localidade de dados, culminando em um desempenho superior em comparação aos cenários que não tiveram tal estratégia. O desempenho obtido pela estratégia de particionamento representou uma melhoria de 410% em relação ao sequencial. A análise de complexidade revela comportamentos distintos:

$$T_{\text{sequencial}} = O(n \times k) \tag{1}$$

$$T_{\text{scheduler}} = O(n \times k + \text{overhead}_{\text{sync}}) \tag{2}$$

$$T_{\text{particionamento}} = O\left(\frac{n \times k}{p}\right) \tag{3}$$

onde n representa o número de documentos, k o número de etapas do pipeline, e p o número de processadores disponíveis. O *overhead* de sincronização observado na estratégia de *scheduler* pode ser quantificado como aproximadamente 5% do tempo total de execução, equivalente a 240 ms para o dataset utilizado.

4.2. Discussão dos Resultados

Os resultados obtidos validam empiricamente os princípios da Lei de Amdahl. De acordo com esses princípios:

1. Para a estratégia de scheduler, a fração paralelizável $P \approx 0\%$ devido às dependências sequenciais, resultando em speedup teórico máximo de 1x. O speedup observado de 0,95x confirma que o overhead supera os benefícios potenciais.
2. Para o particionamento de dados, $P \approx 80{-}85\%$, resultando em speedup teórico de aproximadamente 5,3x para 8 cores, consistente com o valor observado de 5,10x.

Os resultados obtidos demonstram que pipelines NLP que incorporam dependências sequenciais estritas não apresentam ganhos de desempenho significativos quando submetidos a estratégias de paralelização baseada em tarefas. Em contrapartida, a aplicação de técnicas de paralelização por dados revela-se altamente eficaz, proporcionando ganhos substanciais de performance nesses workloads.

5. Considerações Finais

Este trabalho apresentou o desenvolvimento e a avaliação de um pipeline robusto e modular para o pré-processamento de grandes volumes de dados textuais, com o objetivo de otimizar a alimentação de LLMs em tarefas de domínio específico, como a sumarização de documentos jurídicos. A abordagem centralizou-se na representação das tarefas de pré-processamento como um *Directed Acyclic Graph* (DAG) e no escalonamento eficiente dessas tarefas por meio de uma fila de prioridades e execução paralela em C++.

Este estudo contribui para o entendimento de como otimizar pré-processamento textual em larga escala, fornecendo evidência empírica de que:

- Dependências sequenciais limitam drasticamente o paralelismo efetivo;
- *Overhead* de sincronização pode superar benefícios da paralelização;
- Particionamento de dados é eficaz para *workloads* de pré-processamento textual;
- Validação de consistência é fundamental para garantir correção dos resultados.

Também foram identificadas limitações no estudo, inerentes ao escopo da pesquisa, principalmente na utilização de hardware com capacidade de processamento limitada a 8 núcleos e a etapas do pré-processamento simuladas devido ao escopo. Sobretudo, tais considerações refletem a necessidade de explorar cenários futuros considerando tratar as limitações mencionadas.

Sendo assim, a principal contribuição deste trabalho reside na validação de uma arquitetura de pré-processamento de texto que combina um gerenciamento de tarefas baseado em grafo com um agendador que usa fila de prioridades para otimizando o fluxo de trabalho em tarefas de LLMs. O projeto serve como um estudo de caso para o desenvolvimento de *pipelines* robustos e escaláveis, essenciais para o avanço da Ciência de Dados e da Inteligência Artificial Generativa em ambientes de produção. Toda a implementação está disponível neste repositório do github chamado Graph Priority.

Referências

Fu et al. 2024 Fu, Y., Zhu, S., Su, R., Qiao, A., Stoica, I., and Zhang, H. (2024). Efficient LLM scheduling by learning to rank. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Lee et al. 2021 Lee, H., Cho, S., Jang, Y., Lee, J., and Woo, H. (2021). A global dag task scheduler using deep reinforcement learning and graph convolution network. *IEEE Access*, 9:158548–158561.

Mitzenmacher and Shahout 2025 Mitzenmacher, M. and Shahout, R. (2025). Queueing, predictions, and llms: Challenges and open problems. https://arxiv.org/abs/2503.07545.

Ren 2024 Ren, E. (2024). Task scheduling for decentralized llm serving in heterogeneous networks.

Verhaeghe et al. 2024 Verhaeghe, H., Cappart, Q., Pesant, G., and Quimper, C.-G. (2024). Learning precedences for scheduling problems with graph neural networks. In *CP*, pages 30:1–30:18.

Wei et al. 2025 Wei, X., Zhang, J., Li, H., Chen, J., Qu, R., Li, M., Chen, X., and Luo, G. (2025). Agent.xpu: Efficient scheduling of agentic llm workloads on heterogeneous soc.