



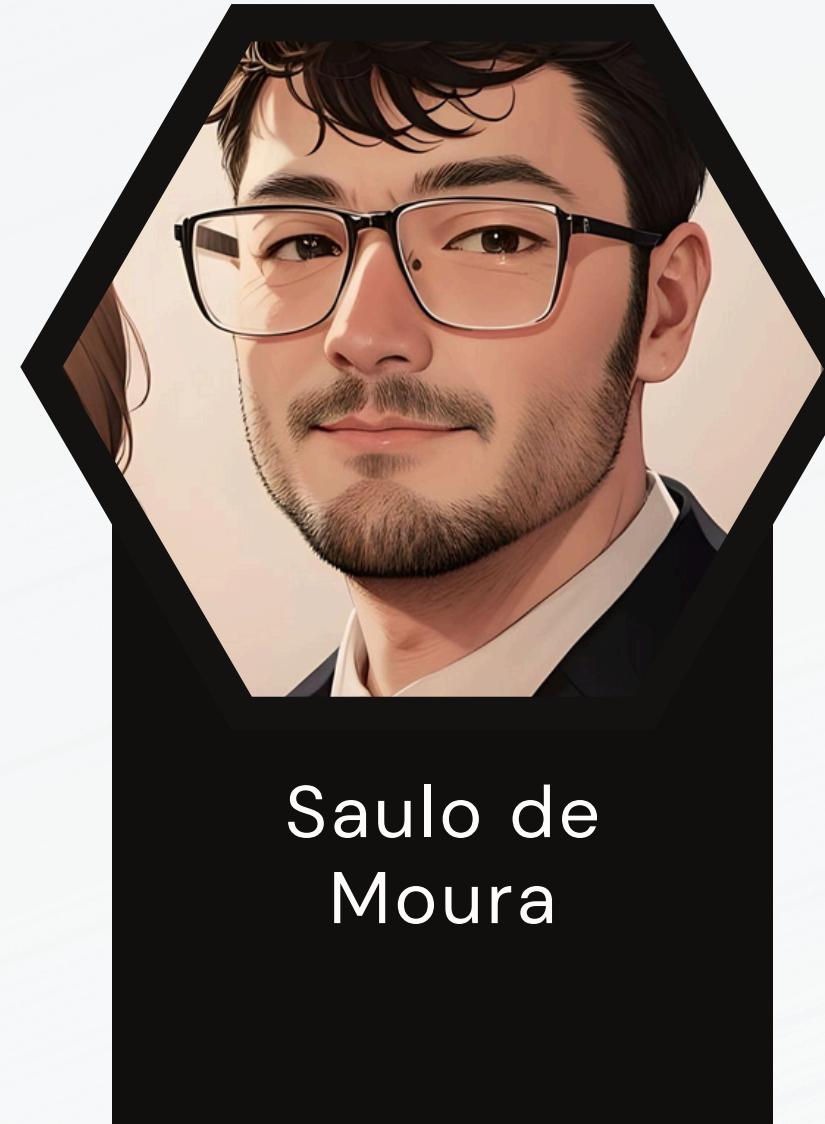
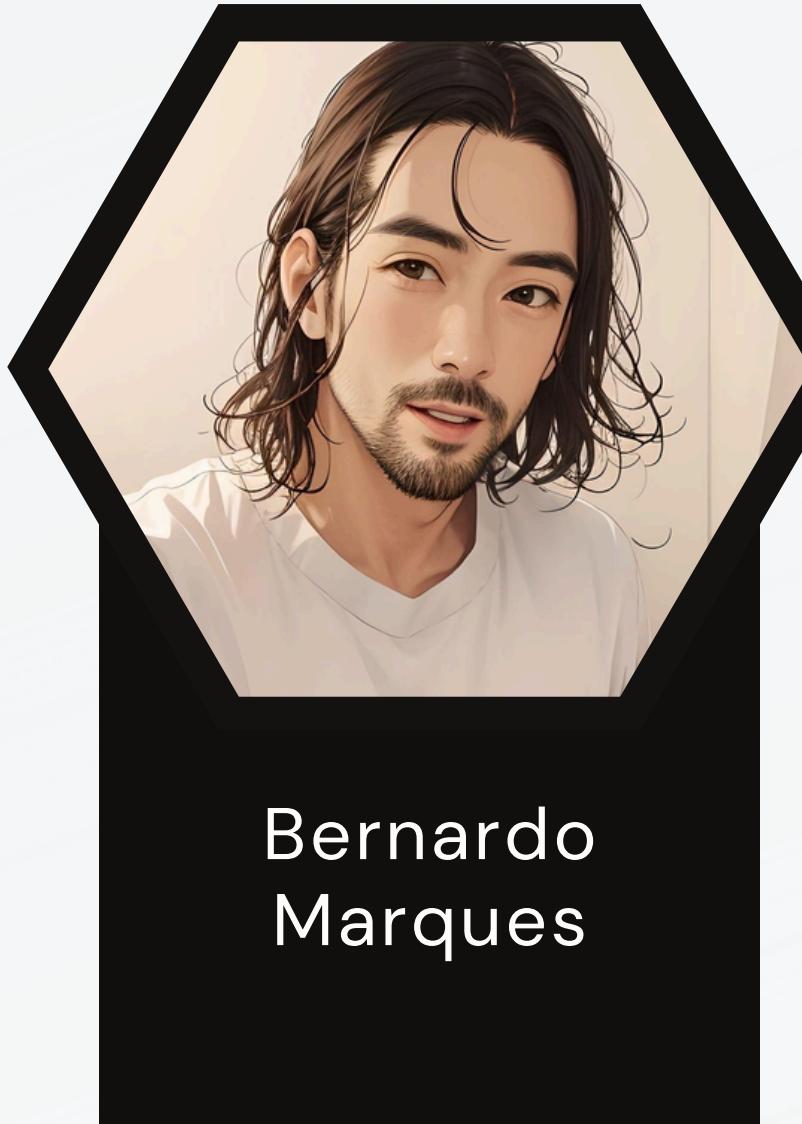
# **PROJETO 1**

# **PARALELIZANDO**

# **UMA IA**



# OUR TEAM



# CONTENT

- 
- 01** RANDOM FOREST
  - 02** PARALELISMO OPENMP
  - 03** SPEEDUPS DO OPENMP
  - 04** OTIMIZAÇÃO COM MPI
  - 05** SPEEDUPS DO MPI
  - 06** DISCUSSÕES E CONSIDERAÇÕES

# MÁQUINA USADA

Sistema operacional: linux mint 21.1

CPU: QUAD CORE MODEL-intel i5 - 11300H

GPU: RTX 3050

Memória RAM: 16GB

# RANDOM FOREST

mais que um Random Forest, uma Floresta aleatória

# RANDOM FOREST

## Cart tree

Árvore de decisão que usa o coeficiente Gini para determinar a melhor característica para a divisão da árvore

## bootstrap sample

Para cada árvore, gera-se uma subpartição das instâncias de treino para que nenhuma árvore tenha 100% a mesma estrutura

## Classificação

O resultado final da predição de cada instância de teste é tirado a partir da votação das previsões locais de cada árvore da floresta



# OPENMP



- Para a versão do código em paralelo com OPENMP utilizamos
- `#pragma omp parallel for schedule(dynamic)`
- `#pragma omp critical`

- Ao parallelizar a função `best_threshold` da classe CART, temos que a criação de cada árvore terá seu tempo de execução reduzido, pois a busca pela melhor regra demanda bastante tempo de execução
- A seção crítica garante que as variáveis `lowest_impurity`, `best_feature` e `best_threshold` não sejam acessadas por mais de uma thread ao mesmo tempo



# SPEEDUP

OPENMP - 1 Thread

Sequencial - 4:06

OpenMP - 4:11

0,9801

# SPEEDUP

OPENMP - 2 threads

Sequencial - 4:06

OpenMP - 2:27

**1,6735**

# SPEEDUP

OPENMP - 4 threads

Sequencial - 4:06

OpenMP - 1:34

2,6170

# SPEEDUP

OPENMP - 8 threads

Sequencial - 4:06

OpenMP - 1:27

2,8276

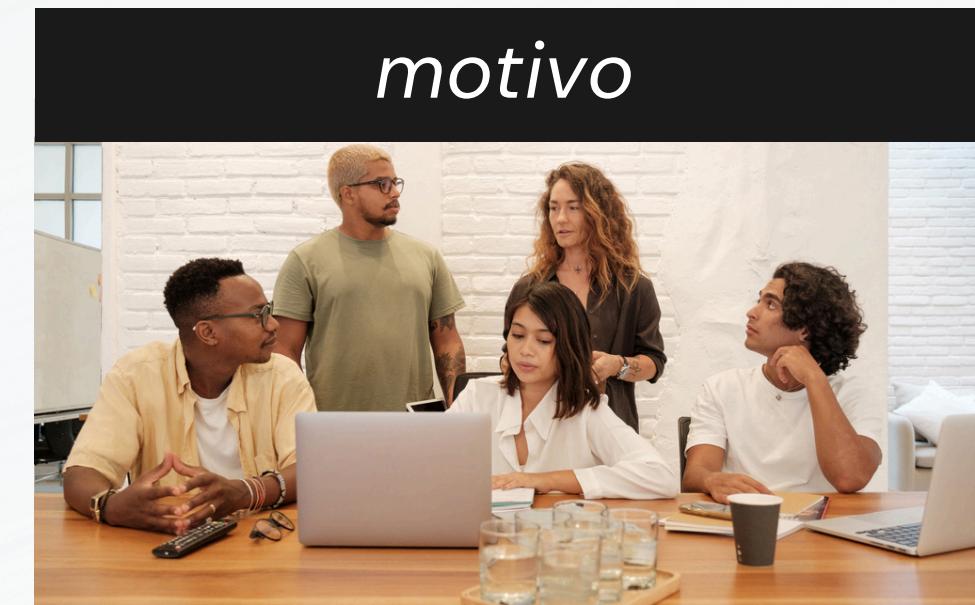
# MPI



*Random Forest*

- Para cada processo, as árvores do random forest são criados, treinados e testados dentro de cada processo. Apenas após o conseguir resultado regional(melhor resultado de cada processo), o processo principal recebe os resultados regionais para descobrir o melhor resultado
- a seção de OPENMP segue a mesma

- Separando grupos de árvores para cada processo facilita a criação, treino e teste das mesmas, assim cada processo trabalha individualmente para realizar suas tarefas e apenas no final temos a junção dos resultados, minimizando o tempo de execução do algoritmo



*motivo*

# SPEEDUP

OPENMP + MPI

1 processo e 4 threads

Sequencial - 4:06

MPI & OpenMP - 4:03

**1,0124**

# SPEEDUP

OPENMP + MPI

2 processos e 2 threads

Sequencial - 4:06

MPI & OpenMP - 2:09

**1,9069**

# SPEEDUP

OPENMP + MPI

4 processos e sem threads

Sequencial - 4:06

MPI & OpenMP - 1:41

**2,4356**



# APRESENTAÇÃO PRÁTICA

# CONSIDERAÇÕES

Com esse Projeto, podemos ver como funciona a integração do OPENMP com o MPI mostrando como essas duas ferramentas trabalham muio bem juntas. Além de visualizarmos na prática como funciona a escalabilidade forte

A pequena melhora entre o teste de 4 threads para o teste com 8 threads

OPENMP

Baixo speedup com 1 processo e 4 threads

Um otimo aumento utilizando apenas MPI

MPI+OPENMP

Mesmo que os ganhos com MPI e OPENMP sejam evidentes, para o nosso problema é visível que a melhor situação é colocar 4 threads

MELHOR  
RESULTADO

**THANK'S FOR  
WATCHING**

