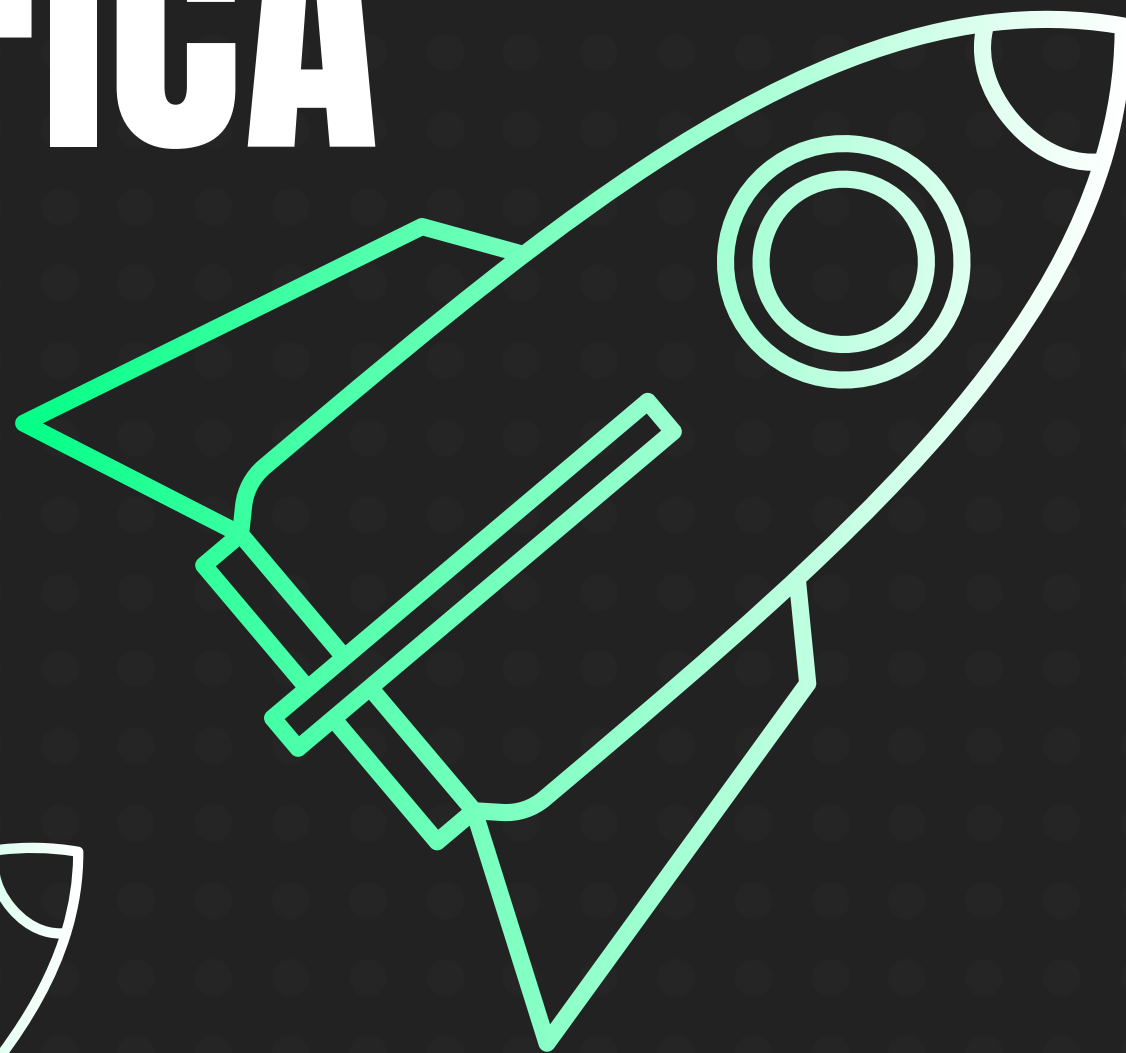


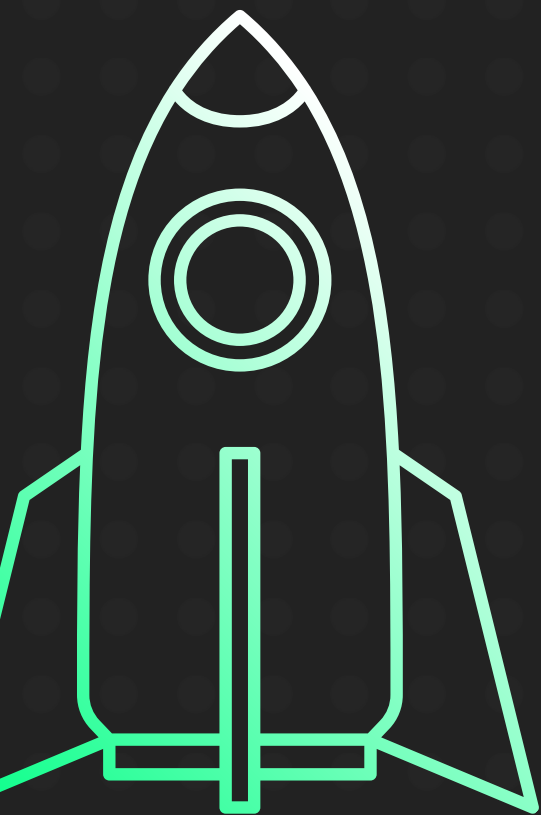
# SPRINT 4

# COMPUTAÇÃO

# NEUROMÓRFICA



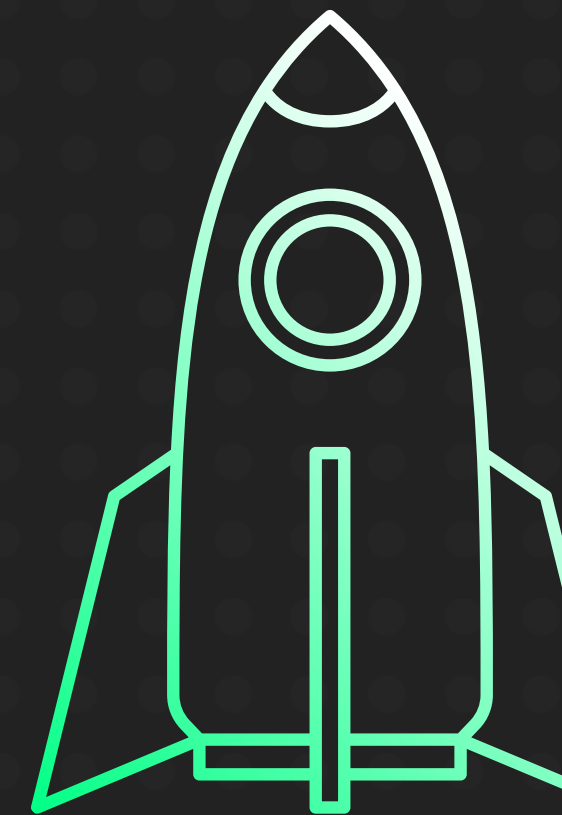
**HENRICO BELA**  
**EMILLY GABRIELLY**  
**SARA LEAL**  
**FELYPE NUNES**  
**DANIEL FARIA**



# Sobre a performance da API

Analizando a performance da API entregue no sprint 3, foi identificado que a mesma necessitava de um upgrade em seu endpoint de “Ingest” onde foi colocada a função de gravar os dados de um endpoint publico (ThingsPeaks).

Foram executadas algumas mudanças neste endpoint de Ingest, e obtivemos os seguintes Benchmarks de Resposta temporal para a nova API.



# Slow Ingest

Conforme á imagem a baixo, chamamos o endpoint antigo de slow\_ingest, onde o codigo necessitava de otimização. Foi calculado o tempo de requisição, e obteve-se o seguinte resultado:

API deployada em Kubernetes

POST 10.1.219.153:5000/slow\_ingest

Params Authorization Headers (9) Body Pre-request Script Tests Settings

Query Params

Key	Value	Description
Key	Value	Description

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2   "message": "Insercao de dados realizada com sucesso, não utilizando tecnicas de otimização",
3   "processing_time_seconds": 1.113792896270752
4 }
```

Status: 200 OK Time: 1120 ms

API deployada em Docker

POST http://172.17.0.2:5000/slow\_ingest

Params Authorization Headers (9) Body Pre-request Script Tests Settings

Query Params

Key	Value	Description
Key	Value	Description

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2   "message": "Insercao de dados realizada com sucesso, não utilizando tecnicas de otimização",
3   "processing_time_seconds": 1.039721965789795
4 }
```

Status: 200 OK Time: 1044 ms

# Fast Ingest

Conforme á imagem a baixo, agora com as otimizações de multithreading, obtivemos resultados satisfatórios nas requisições!

## API deployada em Kubernetes

POST 10.1.219.153:5000/fast\_ingest

Params Authorization Headers (9) Body Pre-request Script Tests Settings

Query Params

Key	Value	Description
Key	Value	Description

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2   "message": "Insercao de dados realizada com sucesso, utilizando Multithreading",
3   "processing_time_seconds": 0.9783880710601807
4 }
```

Status: 200 OK Time: 984 ms

## API deployada em Docker

POST http://172.17.0.2:5000/fast\_ingest

Params Authorization Headers (9) Body Pre-request Script Tests Settings

Query Params

Key	Value	Description
Key	Value	Description

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2   "message": "Insercao de dados realizada com sucesso, utilizando Multithreading",
3   "processing_time_seconds": 1.0312516689300537
4 }
```

Status: 200 OK Time: 1041 ms

## RESULTADOS

Com o calculo de tempo e benchmark entre as funções otimizadas e não otimizadas, ambos nos ambientes de Kubernetes, pode-se ver uma melhora de mais de 10% no tempo da requisição.

Ganho de performance  
**12% – tempo**

Apesar de parecer pouco ganho de performance, estamos falando de uma base de dados pequena, porém, com uma base de dados maior, esta porcentagem tende a aumentar mais ainda, ou seja, o tempo ainda menor utilizando as otimizações.

# Funções Alteradas e Commits

Conforme dito anteriormente, 2 funções foram alteradas para que esta otimização funcionasse.

```
# Função para inserir dados no banco
def inserir_dados():
    conn = sqlite3.connect('api/core/db/data.db')
    cursor = conn.cursor()

    get_df = TksRequest()
    df = get_df.get_data_response()
    colunas = ', '.join(df.columns)
    tabela = "configurations"
    placeholders = ', '.join(['?'] * len(df.columns))

    sql = f'CREATE TABLE IF NOT EXISTS {tabela} (id INTEGER PRIMARY KEY AUTOINCREMENT, {colunas})'
    cursor.execute(sql)
    conn.commit()

    sql = f'INSERT INTO {tabela} ({colunas}) VALUES ({placeholders})'
    values = [tuple(x) for x in df.values.tolist()]
    cursor.execute(f"DELETE FROM {tabela}")
    cursor.executemany(sql, values)
    conn.commit()

    cursor.close()
    conn.close()
```

Repositório GITHUB:

<https://github.com/henricobela/api-ingest>

```
# Aplicando conceito de multithreading
@bp.route('/fast_ingest', methods=['POST'])
def fast_ingest():
    start_time = time.time()

    num_threads = 4
    threads = []

    for _ in range(num_threads):
        thread = threading.Thread(target=inserir_dados)
        threads.append(thread)
        thread.start()

    for thread in threads:
        thread.join()

    end_time = time.time()
    processing_time = end_time - start_time

    return jsonify({'message': 'Insercao de dados realizada com sucesso, utilizando Multithreading',
                    'processing_time_seconds': processing_time})
```

## Commits

Commits on Oct 23, 2023

### Benchmark



henricobela committed 36 minutes ago

### Benchmark



henricobela committed 37 minutes ago

### Multithreading



henricobela committed 48 minutes ago

**MUITO  
OBRIGADO!**