Desenho do Experimento

Introdução

O experimento tem como finalidade comparar o desempenho entre chamadas realizadas via API REST e chamadas realizadas via API GraphQL em repositórios do GitHub. A análise envolve dois indicadores principais: tempo de resposta e tamanho do payload. A seguir, apresenta-se o desenho experimental estruturado em blocos claros e objetivos.

A. Hipóteses

Hipótese Nula (H_0)

Não existe diferença significativa entre REST e GraphQL quanto ao tempo de resposta e ao tamanho do payload.

Hipótese Alternativa (H_1)

A API GraphQL apresenta desempenho superior ao REST, produzindo tempos menores e/ou payload reduzido.

B. Variáveis do Experimento

1. Variáveis Dependentes (mensuradas)

- Tempo de resposta: intervalo entre o envio da requisição e o recebimento da resposta completa.

- Tamanho do payload: quantidade de bytes retornados no corpo da resposta.

2. Variável Independente (manipulada)

- Tipo de API, dividida em duas condições:

    - Condição A: chamadas via REST API

- Condição B: chamadas via GraphQL API

C. Tratamentos

Cada tratamento representa um tipo de consulta aplicada aos repositórios estudados.

Tratamento 1 — REST API

- Endpoint:
  `GET https://api.github.com/repos/{owner}/{repo}`

- Descrição: Retorna metadados do repositório, incluindo o número de issues. Medem-se tempo de resposta e tamanho do payload JSON retornado.

Tratamento 2 — GraphQL API

- Endpoint:
  `POST https://api.github.com/graphql`
  Headers: `Content-Type: application/json`
  Body:

```
{
  "query": "
    query($owner:String!, $repo:String!) {
      repository(owner:$owner, name:$repo) {
        issues(first:100) { totalCount }
      }
    }
  ",
  "variables": { "owner": "{owner}", "repo": "{repo}" }
}
```

Descrição:Consulta o total de *issues* em uma única requisição POST, registrando tempo e tamanho do *payload*.

D. Objetos Experimentais

- Conjunto de repositórios GitHub definidos previamente (por exemplo, dez repositórios populares de diferentes áreas).

- Cada repositório é testado nas duas condições (REST e GraphQL), garantindo um desenho pareado.

E. Tipo de Desenho Experimental

Desenho Pareado (Within-Subjects / Repeated Measures)

1. Cada repositório é analisado sob ambas as condições de API.

2. Diferenças intrínsecas entre repositórios são controladas, porque cada repositório participa dos dois tratamentos.

3. São executadas medições repetidas, alternando a ordem das chamadas para evitar viés de cache ou de sequência.

F. Quantidade de Medições

- Para cada repositório e para cada tratamento, realizam-se 30 medições independentes de tempo e tamanho do payload.

- Total de medições:
  número_de_repositórios × 2 tratamentos × 30 repetições.

- O número 30 segue boas práticas estatísticas, aproximando a distribuição da normalidade.

G. Ameaças à Validade

1. Validade Interna

- Variação de rede: oscilações podem alterar o tempo medido.
  Mitigação: alternância aleatória entre REST e GraphQL, rede estável e descarte de outliers (>3× desvio padrão).

- Caching client-side ou server-side: pode reduzir artificialmente o tempo.
  Mitigação: uso de cabeçalhos `Cache-Control: no-cache` e espera de 1 segundo entre requisições.

2. Validade Externa

- Generalização limitada: repositórios populares não representam o comportamento em repositórios privados ou pequenos.
  Mitigação: selecionar casos variados e, se possível, incluir repositórios menos populares.

3. Validade de Conclusão

- Erro de medição: imprecisões na coleta dos timestamps.
  Mitigação: uso de timestamps em milissegundos e armazenamento dos dados brutos.

- Variação por compressão/codificação: pode alterar o tamanho do payload.
  Mitigação: medir o tamanho exato do corpo JSON.

4. Validade de Construct

- Definição de eficiência: tempo e tamanho podem apontar resultados divergentes.
  Mitigação: análise separada ou cálculo de índice composto (ex.: tempo × tamanho).

5. Validade Social/Ecológica

- Condições artificiais: execução single-thread pode não refletir o uso real.
  Mitigação: registrar essa limitação e sugerir testes futuros com carga concorrente.

Procedimento Detalhado

1. Preparação do Ambiente

a) Criar arquivo `repos_list.csv` com colunas `owner` e `repo`.
b) Exportar variável de ambiente `GITHUB_TOKEN`.
c) Configurar parâmetros no script `experiment.py`:

- `GITHUB_TOKEN = os.getenv("GITHUB_TOKEN")`
- `REPOS_CSV = "repos_list.csv"`
- `TRIALS = 30`
- `OUTPUT_RESULTS = "experiment_results.csv"`

2. Execução do Script (`experiment.py`)

Para cada repositório:

a) Sortear ordem inicial (REST ou GraphQL).

b) REST API:

- Registrar timestamp inicial
- Enviar GET
- Medir tempo
- Registrar tamanho do payload
- Gravar linha:
  `owner, repo, treatment, time_seconds, payload_bytes`

c) Aguardar 1 segundo

d) GraphQL API:

- Montar query

- Registrar timestamp
- Enviar POST
- Medir tempo
- Registrar tamanho
- Gravar linha com tratamento = GraphQL

e) Repetir o processo 30 vezes.

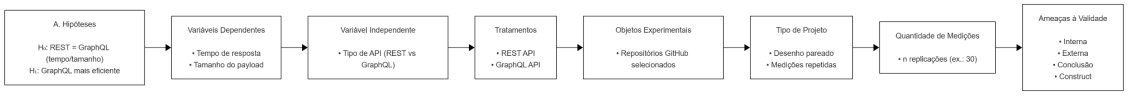
3. Análise Estatística (`analysis.py`)

a) Carregar `experiment_results.csv` em um DataFrame.
b) Agrupar por (`owner, repo, treatment`) e calcular:

- média e desvio padrão de tempo
- média e desvio padrão do payload
  c) Salvar como `experiment_summary.csv`.

4. Visualizações (`dashboard.py`)

a) Ler `experiment_summary.csv`.
b) Gerar histogramas comparando REST e GraphQL para:

- tempos
- tamanhos do payload
  c) Salvar as figuras em PNG ou produzir dashboard interativo
- 

.