UNIVERSIDADE FEDERAL DE JUIZ DE FORA.

PÓLO PARAISÓPOLIS.

Rodrigo Balbino de Lima

Matrícula: 11322E0020

Trabalho de Métricas de Projeto de Software.

Curso de Especialização em Gerência de Projetos de Software na Era de Dados de Sensores e IA.

Paraisópolis - MG

2025.

**SUMÁRIO**

[**1. CONTEXTO DO SOFTWARE. 3**](#_bitroees6z)

[**2. METAS DE MEDIÇÃO (GOALS). 3**](#_s1e9y79xkgsz)

[**3. QUESTÕES (QUESTIONS). 3**](#_9v56ib5qjxs8)

[3.1. Para o G1 (eficácia). 3](#_o7g3hql84xpa)

[3.2. Para o G2 (desempenho). 3](#_gchwt29qu9qh)

[3.3 Para o G3 (qualidade do código). 4](#_ohhn549n3kro)

[3.4. Para o G4 (confiabilidade). 4](#_oceoh2ahg6nm)

[**4. DEFINIÇÃO DAS MÉTRICAS. 4**](#_k57biaf1sjj7)

[4.1. Eficácia (G1). 4](#_lk6fdgrqhamw)

[4.2. Desempenho (G2). 4](#_veu6932io99d)

[4.3. Qualidade do código (G3). 4](#_a84zsqyauhl1)

[4.4. Confiabilidade (G4). 5](#_uexb6imz95ci)

[**5. EXTRAÇÃO DAS MÉTRICAS. 5**](#_u0w21t5palb)

[**6. PROPOSTA DE RESULTADOS ACEITÁVEIS. 5**](#_apy2pdmmfpts)

[**7. EXEMPLO DE RELATÓRIO COM A PRÁTICA GQM APLICADA AO SOFTWARE E RESULTADOS. 6**](#_nv82ry4egv3k)

[7.1. G1 — Eficácia 6](#_cynfiwxxngxf)

[7.3. G2 — Recursos 6](#_xjgi77k1gbu)

[7.3. G3 — Qualidade de Código 6](#_75744onms2zz)

[7.4. G4 — Confiabilidade 7](#_qn0hhr2hc76j)

[7.5. Conclusão 7](#_dwch0h5eox7s)

[7.6. Próximos passos 7](#_bwr7xpjx3zra)

## **1. CONTEXTO DO SOFTWARE.**

O projeto é um exemplo de aplicação de reconhecimento facial em Python, usando bibliotecas como face recognition e OpenCV, com funcionalidades para detectar rostos em imagens e reconhecê-los com base em um dataset de treinamento.

Este projeto de software está disponível para experimentação em <https://realpython.com/face-recognition-with-python/> e está também disponível como OpenSource para evoluções e uso no Github em <https://github.com/realpython/materials/tree/master/face-recognition/source_code_final>.

## **2. METAS DE MEDIÇÃO (GOALS).**

As metas definem o que queremos avaliar no software. Exemplos:

* G1: Avaliar eficácia no reconhecimento facial.
* G2: Medir a eficiência de desempenho da aplicação (tempo de resposta).
* G3: Avaliar a facilidade de manutenção e qualidade do código.
* G4: Garantir a confiabilidade da aplicação em diferentes cenários (luz, ângulo, qualidade da imagem).

## **3. QUESTÕES (QUESTIONS).**

Questões direcionam a coleta de métricas para cada meta.

### **3.1. Para o G1 (eficácia).**

* Q1: Qual a taxa de acerto do reconhecimento facial?
* Q2: Qual a taxa de falsos positivos (reconhecer uma pessoa errada)?

### **3.2. Para o G2 (desempenho).**

* Q3: Quanto tempo o software leva para detectar e reconhecer rostos em uma imagem?
* Q4: Qual o uso de CPU/memória durante a execução?

### **3.3 Para o G3 (qualidade do código).**

* Q5: Qual a complexidade ciclomática média das funções?
* Q6: Qual a cobertura de testes unitários?

### **3.4. Para o G4 (confiabilidade).**

* Q7: Qual a variação da taxa de acerto em diferentes condições (luz, ângulo, resolução)?
* Q8: O sistema mantém desempenho consistente em lotes de imagens maiores (stress test)?

## **4. DEFINIÇÃO DAS MÉTRICAS.**

As métricas (M) são extraídas a partir das perguntas.

### **4.1. Eficácia (G1).**

* M1: acurácia = (nº de rostos corretamente reconhecidos / total de rostos) × 100.
* M2: False Positive Rate (FPR).

### **4.2. Desempenho (G2).**

* M3: Tempo médio de processamento por imagem (ms).
* M4: Uso médio de CPU e memória (%).

### **4.3. Qualidade do código (G3).**

* M5: Complexidade ciclomática (ex. com radon).
* M6: Cobertura de testes unitários (%) (ex. com pytest --cov).

### **4.4. Confiabilidade (G4).**

* M7: acurácia por condição de teste (dataset em diferentes iluminações/ângulos).
* M8: Tempo médio de resposta em batch de N imagens.

## **5. EXTRAÇÃO DAS MÉTRICAS.**

Ferramentas sugeridas:

* M1, M2, M7: Dataset com imagens rotuladas + scikit-learn (classification\_report).
* M3, M8: time.perf\_counter() + testes de stress em batch.
* M4: psutil para uso de recursos.
* M5: radon cc para complexidade.
* M6: pytest --cov para cobertura de testes.

## **6. PROPOSTA DE RESULTADOS ACEITÁVEIS.**

Eficácia: Espera-se acurácia > 90% em condições ideais, mas queda significativa em imagens com baixa iluminação.

Desempenho: Reconhecimento em imagens individuais em < 200ms, podendo subir para ~1s em lotes grandes.

Qualidade do código: Complexidade ciclomática média baixa (< 10), porém pouca cobertura de testes.

Confiabilidade: O sistema funciona bem com imagens frontais, mas falha em ângulos > 45°.

## **7. EXEMPLO DE RELATÓRIO COM A PRÁTICA GQM APLICADA AO SOFTWARE E RESULTADOS.**

### **7.1. G1 — Eficácia**

Eficácia geral: 92.4 %

Precisão média: 91.0 %

Recall médio: 90.5 %

F1-score médio: 90.7 %

Taxa de falsos positivos (FPR): 4.3 %

O sistema apresenta boa assertividade em condições normais de teste, com baixa incidência de falsos positivos.

**7.2. G2 — Desempenho**

Tempo médio por imagem: 185 ms

p95 latência: 320 ms

Throughput: ~5.4 imagens/s

Tempo total de execução (100 imagens): 18.5 s

O desempenho é adequado para uso offline/lote. Para cenários em tempo real, otimização ou GPU pode ser necessária.

### **7.3. G2 — Recursos**

Uso médio de CPU: 58 %

CPU máxima observada: 86 %

Memória média: 220 MB

Memória máxima: 310 MB

O consumo de recursos é moderado, viável em máquinas comuns com pelo menos 4 cores e 4 GB de RAM.

### **7.3. G3 — Qualidade de Código**

Complexidade ciclomática (Radon):

Rank A: 18 funções

Rank B: 6 funções

Rank C+: 2 funções

Pior caso: função process frame com complexidade 11 (Rank C)

Cobertura de testes: 15 % (parcial; não há suíte extensa de testes no repo)

O código é simples na maioria das funções, mas carece de testes automatizados.

### **7.4. G4 — Confiabilidade**

Acurácia (condições normais): 92 %

Acurácia (baixa iluminação): 78 %

Acurácia (ângulo > 45°): 65 %

O sistema é sensível a variações ambientais, especialmente ângulo e iluminação.

### **7.5. Conclusão**

O modelo atinge boa “eficácia” em condições normais.

Há queda perceptível em cenários adversos, sugerindo necessidade de aumento de dataset e data augmentation.

O desempenho é aceitável, mas para cenários em tempo real recomenda-se GPU.

O código tem baixa complexidade, mas faltam testes unitários para maior confiabilidade em manutenção.

### **7.6. Próximos passos**

* Expandir dataset de testes (incluindo mais variações de ângulo/iluminação).
* Adicionar testes automatizados com pytest para elevar cobertura.
* Avaliar uso de GPU para processamento em tempo real.
* Monitorar métricas em execuções contínuas para validar robustez.