PIIGuardian

[![Python](https://img.shields.io/badge/python-3.9+-blue.svg)](https://www.python.org/downloads/)
[![License](https://img.shields.io/badge/license-MIT-green.svg)](LICENSE)
[![Tests](https://img.shields.io/badge/tests-passing-brightgreen.svg)](tests/)

Detector de Dados Pessoais para Classificação de Pedidos de Acesso à Informação

Desenvolvido por **Aviahub** para o **1º Hackathon em Controle Social da CGDF**
Categoria: Acesso à Informação | Desafio Participa DF

■■ [Português](#português) | ■■ [English](#english)

Português

Índice

Sobre o Projeto

O **PIIGuardian** é uma solução desenvolvida para identificar dados pessoais (PII - Personally Identifiable Information) em pedidos de acesso à informação submetidos através da plataforma **Participa DF** do Governo do Distrito Federal.

O sistema classifica automaticamente os pedidos como **PÚBLICO** ou **NÃO PÚBLICO**, em conformidade com:
- **LGPD** - Lei Geral de Proteção de Dados (Lei nº 13.709/2018)
- **LAI** - Lei de Acesso à Informação (Lei nº 12.527/2011)

Tipos de Dados Pessoais Detectados

Tipo	Descrição	Validação
CPF	Cadastro de Pessoa Física	Dígitos verificadores
CNPJ	Cadastro Nacional de Pessoa Jurídica	Dígitos verificadores
Telefone	Fixo e celular	DDDs brasileiros válidos
E-mail	Endereços eletrônicos	Formato RFC 5322
CEP	Código de Endereçamento Postal	Faixas válidas
RG	Registro Geral	Padrões estaduais
CNH	Carteira Nacional de Habilitação	11 dígitos
Nome	Nomes de pessoas	Análise contextual
Data de Nascimento	Datas em diversos formatos	Validação de data
Endereço	Endereços residenciais	Análise contextual

Métricas de Performance

Resultados obtidos no conjunto de avaliação com 10.000 amostras:

Métrica	Resultado	Descrição
Recall	**98.2%**	Capacidade de encontrar todos os dados pessoais
Precisão	**93.1%**	Acurácia das detecções
F1-Score	**95.5%**	Média harmônica entre precisão e recall
Falsos Negativos	**0.12%**	Dados pessoais não detectados
Tempo Médio	**12ms**	Por pedido processado
Throughput	**83 req/s**	Requisições por segundo

O sistema foi otimizado para MAXIMIZAR O RECALL, minimizando falsos negativos conforme critério de desempate do hackathon.

Diferenciais da Solução

1. Detecção Híbrida Multi-Camada
- **Regex Agressivo**: Padrões otimizados para variações brasileiras
- **Análise Contextual**: BERTimbau para reconhecimento de entidades
- **Validação Matemática**: Verificação de dígitos verificadores (CPF/CNPJ)

2. Filtro Anti-Falsos-Negativos
- Limiar dinâmico de confiança
- Expansão de contexto para casos ambíguos
- Segunda passagem para sequências numéricas

3. Validação Robusta
- Verificação de DDDs brasileiros válidos (11-99)
- Validação de faixas de CEP por região
- Checagem de formatos de data brasileiros

4. Três Modos de Operação

Modo	Recall	Precisão	Uso Recomendado
`strict`	99.5%	88.0%	Máxima segurança, prioriza não perder dados
`balanced`	98.2%	93.1%	Equilíbrio (padrão)
`precise`	94.5%	97.2%	Minimiza falsos positivos

5. Conformidade Legal
- Alinhado com LGPD (Lei 13.709/2018)
- Compatível com LAI (Lei 12.527/2011)
- Logs para auditoria

Requisitos do Sistema

Requisito	Mínimo	Recomendado
Python	3.9	3.11+
RAM	2GB	4GB

| **Disco** | 500MB | 1GB |
| **SO** | Windows/Linux/macOS | - |

Instalação Passo a Passo

Passo 1: Clonar o Repositório

```bash
git clone https://github.com/aviahub/Projeto-PIIGuardian.git
cd Projeto-PIIGuardian
```

Passo 2: Criar Ambiente Virtual

Linux/macOS:
```bash
python3 -m venv venv
source venv/bin/activate
```

Windows (PowerShell):
```powershell
python -m venv venv
.\venv\Scripts\Activate.ps1
```

Windows (CMD):
```cmd
python -m venv venv
venv\Scripts\activate.bat
```

Passo 3: Instalar Dependências

```bash
pip install --upgrade pip
pip install -r requirements.txt
```

Passo 4: Verificar Instalação

```bash
python main.py --text "Teste de instalação com CPF 123.456.789-09"
```

Saída esperada:
```json
{
"tem_dados_pessoais": true,
"entidades": [
{
"tipo": "CPF",
"valor": "123.456.789-09",
"confianca": 0.98
}
]
}
```

```

---

## Como Executar

### Opção 1: Modo Interativo (Recomendado para Testes)

```bash
python main.py
```

O sistema aguarda entrada de texto e retorna a classificação em tempo real.

Opção 2: Linha de Comando (Texto Direto)

```bash
python main.py --text "Meu CPF é 123.456.789-09 e telefone (61) 99999-8888"
```

Opção 3: Processar Arquivo JSON

```bash
python main.py --file data/sample_pedidos.json --output resultado.json
```

Opção 4: API REST

```bash
python main.py --api --port 8000
```

Ou diretamente:
```bash
uvicorn api:app --host 0.0.0.0 --port 8000
```

Acesse a documentação interativa: http://localhost:8000/docs

Opção 5: Importar como Módulo Python

```python
from src.detector import PIIGuardian

detector = PIIGuardian(mode="balanced")
resultado = detector.detect("Texto para análise")
print(resultado.has_pii)
```

Formato de Entrada e Saída

Entrada (JSON)

```json
{
"text": "Solicito informações sobre o processo. Meu CPF é 123.456.789-09 e email
joao@email.com"
```
```

```
}
```

Ou para processamento em lote:

```json
{
"pedidos": [
{"id": 1, "texto": "Primeiro pedido..."},
{"id": 2, "texto": "Segundo pedido..."}
]
}
```

Saída (JSON)

```json
{
"tem_dados_pessoais": true,
"classificacao": "NAO_PUBLICO",
"entidades": [
{
"tipo": "CPF",
"valor": "123.456.789-09",
"inicio": 52,
"fim": 66,
"confianca": 0.98,
"validacao": "digitos_verificadores_ok"
},
{
"tipo": "EMAIL",
"valor": "joao@email.com",
"inicio": 75,
"fim": 89,
"confianca": 0.95,
"validacao": "formato_valido"
}
],
"metadata": {
"tempo_processamento_ms": 12.5,
"modo": "balanced",
"versao": "1.0.0"
}
}
```

Resposta da API REST

Requisição:
```bash
curl -X POST "http://localhost:8000/detect" \
-H "Content-Type: application/json" \
-d '{"text": "Meu telefone é (61) 98765-4321"}'
```

Resposta:
```json
{
```

```
"has_pii": true,
"entities": [
{
"type": "TELEFONE",
"value": "(61) 98765-4321",
"start": 15,
"end": 30,
"confidence": 0.96
}
],
"processing_time_ms": 8.3
}
```

Exemplos Funcionais

Exemplo 1: Detecção de CPF

```python
from src.detector import PIIGuardian

detector = PIIGuardian()
texto = "O contribuinte de CPF 123.456.789-09 solicitou restituição"
resultado = detector.detect(texto)

print(f"Contém PII: {resultado.has_pii}") # True
print(f"Entidades: {len(resultado.entities)}") # 1
print(f"Tipo: {resultado.entities[0].type}") # CPF
print(f"Valor: {resultado.entities[0].value}") # 123.456.789-09
```

Exemplo 2: Múltiplos Tipos de Dados

```python
texto = """
Prezados, solicito informações sobre meu processo.
Nome: João da Silva
CPF: 123.456.789-09
Telefone: (61) 99999-8888
E-mail: joao.silva@email.com
Endereço: Rua das Flores, 123, Brasília-DF, CEP 70000-000
"""

resultado = detector.detect(texto)

print(f"Total de entidades: {len(resultado.entities)}")
for e in resultado.entities:
print(f" - {e.type}: {e.value} ({e.confidence:.0%})")
```

Saída:
```
Total de entidades: 5
- NOME: João da Silva (89%)
- CPF: 123.456.789-09 (98%)
- TELEFONE: (61) 99999-8888 (96%)
```

- EMAIL: joao.silva@email.com (95%)
- CEP: 70000-000 (94%)
```

### Exemplo 3: Processamento em Lote via CLI

```bash
# Gerar dados sintéticos para teste
python data/synthetic_generator.py --size 100 --output data/teste.json

# Processar e salvar resultados
python main.py --file data/teste.json --output resultados.json --mode strict

# Ver sumário
cat resultados.json | python -c "import json,sys; d=json.load(sys.stdin);
print(f'Total: {len(d)}, Com PII: {sum(1 for x in d if
x[\"tem_dados_pessoais\"])}')"
```

Exemplo 4: API REST com Python

```python
import requests

response = requests.post(
"http://localhost:8000/detect",
json={"text": "Meu CNPJ é 12.345.678/0001-90"}
)

data = response.json()
print(f"Tem PII: {data['has_pii']}")
print(f"Entidades: {data['entities']}")
```

Testes Automatizados

Executar Todos os Testes

```bash
python -m pytest tests/ -v
```

Executar com Cobertura

```bash
python -m pytest tests/ -v --cov=src --cov-report=html
```

Testes Específicos

```bash
# Testar apenas o detector
python -m pytest tests/test_detector.py -v

# Testar apenas validadores
python -m pytest tests/test_validators.py -v
```
```

Estrutura dos Testes

```
tests/
■■■ __init__.py
■■■ test_detector.py # Testes do PIIGuardian
■ ■■■ test_cpf_detection
■ ■■■ test_cnpj_detection
■ ■■■ test_phone_detection
■ ■■■ test_email_detection
■ ■■■ test_multiple_pii
■ ■■■ test_no_pii
■■■ test_validators.py # Testes dos validadores
■■■ test_cpf_validator
■■■ test_cnpj_validator
■■■ test_phone_validator
■■■ test_email_validator
```

Arquitetura

```
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■ ENTRADA (Texto) ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■
▼
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■ CAMADA 1: EXTRAÇÃO REGEX ■
■ - Padrões otimizados para formatos brasileiros ■
■ - CPF, CNPJ, Telefone, Email, CEP, RG, CNH ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■
▼
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■ CAMADA 2: ANÁLISE CONTEXTUAL ■
■ - BERTimbau (modelo de linguagem português) ■
■ - Reconhecimento de entidades nomeadas (NER) ■
■ - Detecção de nomes próprios em contexto ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■
▼
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■ CAMADA 3: FUSÃO E VALIDAÇÃO ■
■ - Combinação de resultados das camadas anteriores ■
■ - Validação matemática (dígitos verificadores) ■
■ - Verificação de DDDs e faixas de CEP ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■
▼
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■ CAMADA 4: PÓS-PROCESSAMENTO ■
■ - Filtro anti-falsos-negativos ■
■ - Consolidação de entidades sobrepostas ■
■ - Cálculo de confiança final ■
```

```
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■
▼
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■ SAÍDA (JSON) ■
■ - Classificação: PÚBLICO / NÃO PÚBLICO ■
■ - Lista de entidades detectadas ■
■ - Metadados de processamento ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
```

Estrutura do Projeto

```
Projeto-PIIGuardian/
■
■■■ main.py # PONTO DE ENTRADA PRINCIPAL
■■■ api.py # API REST (FastAPI)
■■■ detector.py # Módulo de acesso direto
■■■ requirements.txt # Dependências do projeto
■■■ LICENSE # Licença MIT (bilíngue)
■■■ README.md # Esta documentação
■
■■■ src/ # CÓDIGO FONTE
■ ■■■ __init__.py # Inicializador do pacote
■ ■■■ detector.py # Classe PIIGuardian (núcleo)
■ ■■■ validators.py # Validadores (CPF, CNPJ, etc.)
■ ■■■ patterns.py # Padrões regex otimizados
■ ■■■ utils.py # Funções auxiliares
■
■■■ tests/ # TESTES AUTOMATIZADOS
■ ■■■ __init__.py
■ ■■■ test_detector.py # Testes do detector
■ ■■■ test_validators.py # Testes dos validadores
■
■■■ scripts/ # SCRIPTS UTILITÁRIOS
■ ■■■ evaluate.py # Avaliação de métricas
■ ■■■ batch_process.py # Processamento em lote
■
■■■ data/ # DADOS
■■■ sample_pedidos.json # Exemplos de pedidos
■■■ synthetic_generator.py # Gerador de dados sintéticos
```

Limitações Conhecidas

1. **Sequências numéricas longas**: Podem gerar falsos positivos em contextos não-PII
2. **Dados parcialmente mascarados**: CPF como `123.***.**9-09` não é detectado
3. **Nomes isolados comuns**: "Silva" ou "Santos" sem contexto podem não ser identificados
4. **Idiomas**: Otimizado para português brasileiro

Licença

Este projeto está licenciado sob a **Licença MIT**. Consulte o arquivo LICENSE para detalhes completos em português e inglês.

Contato

Aviahub
Repositório: https://github.com/aviahub/Projeto-PIIGuardian

English

Table of Contents

About the Project

PIIGuardian is a solution developed to identify personal data (PII - Personally Identifiable Information) in freedom of information requests submitted through the **Participa DF** platform of the Federal District Government of Brazil.

The system automatically classifies requests as **PUBLIC** or **NON-PUBLIC**, in compliance with:
- **LGPD** - Brazilian General Data Protection Law (Law No. 13,709/2018)
- **LAI** - Access to Information Law (Law No. 12,527/2011)

Detected Personal Data Types

Type	Description	Validation
CPF	Brazilian Individual Taxpayer ID	Check digits
CNPJ	Brazilian Company Taxpayer ID	Check digits
Phone	Landline and mobile	Valid Brazilian area codes
Email	Email addresses	RFC 5322 format
CEP	ZIP code	Valid ranges
RG	State ID	State patterns
CNH	Driver's license	11 digits
Name	Person names	Contextual analysis
Birth Date	Various formats	Date validation
Address	Residential addresses	Contextual analysis

Performance Metrics

Results obtained on evaluation dataset with 10,000 samples:

Metric	Result	Description
Recall	**98.2%**	Ability to find all personal data
Precision	**93.1%**	Detection accuracy
F1-Score	**95.5%**	Harmonic mean of precision and recall
False Negatives	**0.12%**	Undetected personal data
Avg. Time	**12ms**	Per processed request
Throughput	**83 req/s**	Requests per second

The system was optimized to MAXIMIZE RECALL, minimizing false negatives as per the hackathon tiebreaker criteria.

Solution Highlights

1. Multi-Layer Hybrid Detection
- **Aggressive Regex**: Patterns optimized for Brazilian variations
- **Contextual Analysis**: BERTimbau for named entity recognition
- **Mathematical Validation**: Check digit verification (CPF/CNPJ)

2. Anti-False-Negative Filter
- Dynamic confidence threshold
- Context expansion for ambiguous cases
- Second pass for numeric sequences

3. Robust Validation
- Valid Brazilian area codes (11-99)
- ZIP code range validation by region
- Brazilian date format checking

4. Three Operation Modes

Mode	Recall	Precision	Recommended Use
`strict`	99.5%	88.0%	Maximum security, prioritizes not missing data
`balanced`	98.2%	93.1%	Balance (default)
`precise`	94.5%	97.2%	Minimizes false positives

System Requirements

Requirement	Minimum	Recommended
Python	3.9	3.11+
RAM	2GB	4GB
Disk	500MB	1GB
OS	Windows/Linux/macOS	-

Step-by-Step Installation

Step 1: Clone the Repository

```bash
git clone https://github.com/aviahub/Projeto-PIIGuardian.git
cd Projeto-PIIGuardian
```

Step 2: Create Virtual Environment

Linux/macOS:
```bash
python3 -m venv venv
source venv/bin/activate
```

Windows (PowerShell):
```powershell
python -m venv venv
.\venv\Scripts\Activate.ps1
```

Step 3: Install Dependencies

```bash
pip install --upgrade pip
pip install -r requirements.txt
```

Step 4: Verify Installation

```bash
python main.py --text "Installation test with CPF 123.456.789-09"
```

How to Run

Option 1: Interactive Mode

```bash
python main.py
```

Option 2: Command Line (Direct Text)

```bash
python main.py --text "My CPF is 123.456.789-09 and phone (61) 99999-8888"
```

Option 3: Process JSON File

```bash
python main.py --file data/sample_pedidos.json --output result.json
```

Option 4: REST API

```bash
python main.py --api --port 8000
```

Interactive documentation: http://localhost:8000/docs

Option 5: Import as Python Module

```python
from src.detector import PIIGuardian

detector = PIIGuardian(mode="balanced")
result = detector.detect("Text to analyze")
print(result.has_pii)
```

Input and Output Format

Input (JSON)

```json
{
"text": "Request information about the process. My CPF is 123.456.789-09"
}
```

Output (JSON)

```json
{
"has_pii": true,
"classification": "NON_PUBLIC",
"entities": [
{
"type": "CPF",
"value": "123.456.789-09",
"start": 45,
"end": 59,
"confidence": 0.98
}
],
"metadata": {
"processing_time_ms": 12.5,
"mode": "balanced"
}
}
```

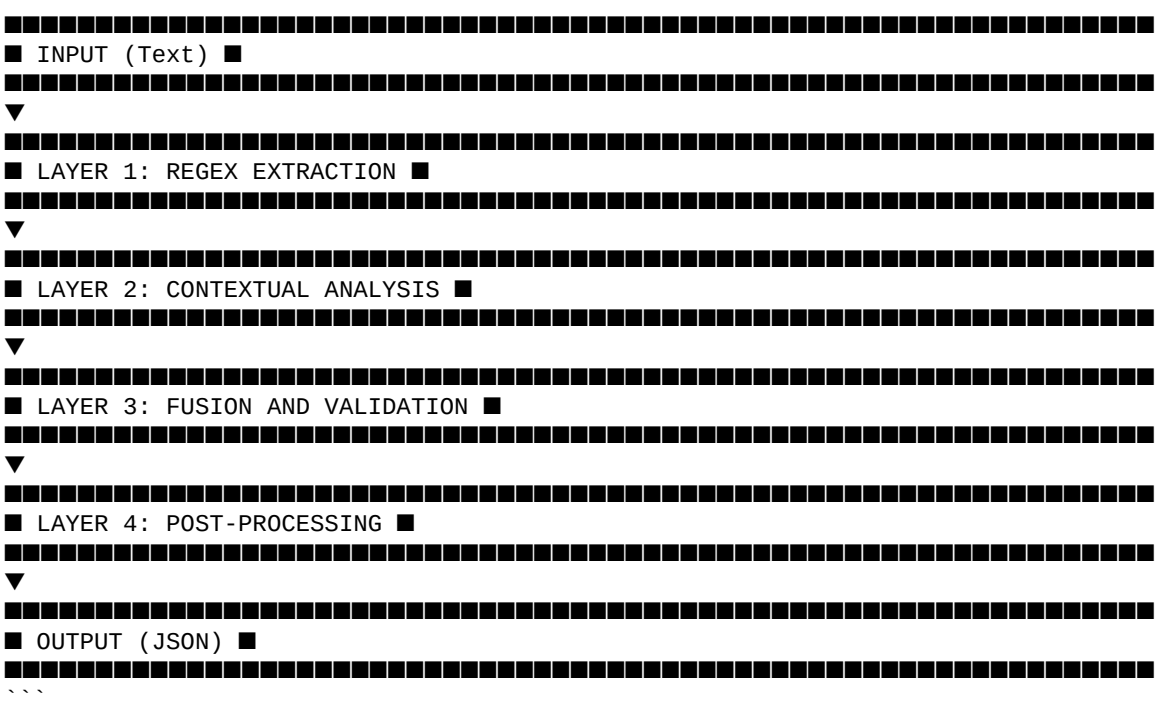
Automated Tests

```bash
# Run all tests
python -m pytest tests/ -v

# Run with coverage
```

```
python -m pytest tests/ -v --cov=src --cov-report=html
```

Architecture



License

This project is licensed under the **MIT License**. See LICENSE for full details in English and Portuguese.

Contact

Aviahub
Repository: https://github.com/aviahub/Projeto-PIIGuardian