Tutorial: Análise Arquitetural do repositório ChatTTS usando DistilBERT, BERT-NER, CodeBERT, com *Flan-T5 Base* e *RoBERTa Sentiment*

GRUPO 5:

01 - Filippi Reis Menezes - 202300027230
02 - Jackson Santana Carvalho Júnior - 202300027365
03- Gabriel Bastos Pimentel - 202300061590
04- Marcos Vinícius Dantas Aguiar - 201800084345
05- Caio Victor Prado Cruz - 202100011234
06- Yan Victor Araujo do Nascimento - 202100046006
07- Leonardo de Souza Aragão - 202200117002
08 - Vênisson Cardoso Dos Santos – 201700063182

Responsáveis pelas análises dos modelos DistilBERT, BERT-NER, CodeBERT:
01 - Filippi Reis Menezes
02- Jackson Santana Carvalho Júnior

Responsáveis pelos modelos DistilBERT, BERT-NER, CodeBERT:
01 - Filippi Reis Menezes
02 - Jackson Santana Carvalho Júnior
03- Gabriel Bastos Pimentel
04- Marcos Vinícius Dantas Aguiar

Responsáveis pelas análises dos modelos *Flan-T5 Base* e *RoBERTa Sentiment:*
06- Yan Victor Araujo do Nascimento
07- Leonardo de Souza Aragão

Responsáveis pelos modelos *Flan-T5 Base* e *RoBERTa Sentiment:*
05- Caio Victor Prado Cruz
06- Yan Victor Araujo do Nascimento
07- Leonardo de Souza Aragão
08 - Vênisson Cardoso Dos Santos

O código fonte de dos tutoriais está localizado aqui:https://github.com/caioprado-dot/Engenharia_SoftwareII_2025-2_T04_ChatTTS

Tutorial: Análise Arquitetural do repositório ChatTTS usando DistilBERT, BERT-NER e CodeBERT

Este README descreve **u**ma analise do repositório **2noise/ChatTTS** para identificar indícios de padrões arquiteturais.
O propósito é **permitir replicação** da atividade por qualquer pessoa, em qualquer IDE ou ambiente (VSCode, PyCharm, Jupyter, terminal), usando os três modelos: **DistilBERT (qualidade/sentimento)**, **BERT-NER (extração de entidades)** e **CodeBERT (análise de código)**.

---

1 — Visão rápida (o que este projeto faz)

1. Clona ou lê o repositório 2noise/ChatTTS.
2. Coleta documentação, arquivos de configuração e arquivos de código.
3. Executa três análises independentes:
    - **DistilBERT** — analisa qualidade/sentimento da documentação (evidências textuais).
    - **BERT-NER** — identifica entidades/termos arquiteturais (services, endpoints, brokers, etc.).
    - **CodeBERT** — analisa trechos de código para inferir características (classes, endpoints, eventos) e sugere padrões.
4. Gera artefatos replicáveis: CSVs, JSON de resumo, gráficos PNG e um relatório PDF.

O objetivo do tutorial abaixo é **explicar como executar essas três análises em qualquer ambiente**.

---

2 — Pré-requisitos (instalação mínima)

Recomendado: Python 3.9+ e git instalado no sistema.

Crie um ambiente virtual e instale dependências:

criar e ativar venv (Linux / macOS)

python -m venv .venv

source .venv/bin/activate

Windows (PowerShell)

python -m venv .venv

.venv\\Scripts\\Activate.ps1

Instalar dependências (recomendado colocar no requirements.txt)

pip install transformers[torch] sentence-transformers requests pandas matplotlib reportlab tqdm scikit-learn torch numpy seaborn

Observação: se você usar GPU, tenha torch instalado com suporte CUDA compatível com sua GPU.

---

3 — Como organizar os arquivos (estrutura mínima recomendada)

Coloque os scripts que você recebeu (as 4 partes) no diretório tools/ com estes nomes (sugestão — não é obrigatório, mas facilita):

ChatTTS/ # repositório local (este README.md fica aqui)

└─ tools/

├─ analyze_repo.py # PARTE 1/4 - coleta + zero-shot + embeddings (opcional)

├─ run_comprehensive_analysis.py # PARTE 2/4 - clone + DistilBERT + BERT-NER + CodeBERT

├─ visualize_results.py # PARTE 3/4 - gera gráficos e CSVs

└─ create_pdf_report.py # PARTE 4/4 - gera o PDF final

- OUTPUT_DIR padrão nos scripts é chattts_analysis (ou chattts_analysis_output em PARTE 1). Pode alterar no topo de cada script.
- Garanta que os scripts importem numpy se referenciam np. (Se receber erro NameError: np, adicione import numpy as np).

---

4 — Modelos: nomes e como apontar para modelos locais ou Hugging Face

Os scripts aceitam (por padrão) os nomes de checkpoints hospedados no Hugging Face. Exemplos usados nos scripts enviados:

- DistilBERT (qualidade / sentimento):
  distilbert/distilbert-base-uncased-finetuned-sst-2-english
- BERT-NER (token classification):
  dslim/bert-base-NER
- CodeBERT (código):
  microsoft/codebert-base (usado com AutoTokenizer e AutoModel)

Você pode usar **paths locais** se tiver modelos baixados:

MODEL_SENTIMENT = "/caminho/para/distilbert_local"

MODEL_NER = "/caminho/para/bert_ner_local"

MODEL_CODE = "/caminho/para/codebert_local"

O transformers.pipeline e AutoModel aceitam diretórios locais que contenham config.json e pesos (pytorch_model.bin ou *.safetensors).

---

5 — Executando (modo terminal / CLI) — passos práticos

5.1 — Passo 1: rodar a análise principal (clone e modelos)

No terminal, estando na raiz do repositório (onde está tools/):

execução do pipeline principal (PARTE 2/4)

python tools/run_comprehensive_analysis.py

- O script clona GITHUB_REPO (padrão: 2noise/ChatTTS) em chattts_analysis/cloned_repo.
- Vai inicializar os modelos (pode baixar pesos — requer internet).
- Saídas em CSV e JSON serão escritas em chattts_analysis/:
  - dataset_completo.csv
  - analise_qualidade.csv
  - entidades_extraidas.csv
  - analise_codigo.csv
  - resumo_analise.json

Se quiser rodar um repositório diferente, altere GITHUB_REPO no topo do script ou modifique para aceitar argumento CLI (ver seção 8 “Dicas para tornar CLI”).

5.2 — Passo 2: gerar visualizações (gráficos) — PARTE 3/4

Depois que os CSVs estiverem prontos:

python tools/visualize_results.py

- O script lê os CSVs e gera PNGs (por exemplo analise_qualidade_detalhada.png, analise_entidades_detalhada.png, analise_codigo_detalhada.png, comparacao_final_modelos.png) em chattts_analysis/.

5.3 — Passo 3: gerar relatório PDF final — PARTE 4/4

Com os PNGs e resumo_analise.json disponíveis:

python tools/create_pdf_report.py

- Gera chattts_analysis/Relatorio_Analise_Arquitetural_ChatTTS.pdf.

---

6 — Rodando em diferentes ambientes / IDEs

Qualquer IDE (VSCode, PyCharm, Thonny, etc.)

1. Abra o projeto no IDE.
2. Configure o interpretador Python (o venv criado).
3. No terminal integrado, rode os comandos indicados (ex.: python tools/run_comprehensive_analysis.py).
4. Se preferir depurar interativamente, abra os scripts e coloque breakpoints.

Jupyter / Colab

- Cole as células (ou converta os scripts em um notebook) e execute célula a célula.
- Atenção: pipelines do transformers fazem download de modelos — se usar Colab, habilite GPU no runtime para acelerar CodeBERT / TF.
- Se usar Colab e tiver limite de RAM, reduza max_files e chame o pipeline para poucos arquivos.

Execução remota / servidor (headless)

- Recomendado usar tmux ou screen para execuções longas.
- Configure CUDA_VISIBLE_DEVICES se houver GPU.
- Monitore logs (imprima status em arquivos de log).

---

7 — Formatos de entrada e saída (para replicação automática)

Entrada

- O pipeline aceita um repositório GitHub (string owner/repo) ou um repositório clonado localmente (quando você adaptar gather_local_files).
- Os scripts processam 3 categorias: documentation (README/MD/TXT), configuration (requirements/Dockerfile), code (.py/.js/.ts/etc.).

Saída (artefatos principais)

- dataset_completo.csv — linhas: path, type, chunk_index, text.
- analise_qualidade.csv — resultados DistilBERT (scores, keywords e qualidade).
- entidades_extraidas.csv — resultados BERT-NER (lista de entidades por chunk).
- analise_codigo.csv — resultados CodeBERT (características e padrões detectados).
- resumo_analise.json — resumo quantitativo consolidado.
- PNGs com gráficos e Relatorio_Analise_Arquitetural_ChatTTS.pdf.

Exemplo de linha (JSON de saída simplificado):


```
{
  "sample_id": "projA",
  "detected_patterns": [
    {"pattern": "Microservices", "confidence": 0.87, "evidence": ["services/* with Dockerfile"]},
    {"pattern": "Event-Driven", "confidence": 0.61, "evidence": ["kafka topics"]}
  ],
  "analysis_meta": {"num_files_scanned": 312, "duration_seconds": 23.4}
}
```


---

8 — Como adaptar para escolher o IDE preferido (resumo)

Você não precisa alterar o código para mudar de IDE. O fluxo é idêntico em qualquer ambiente:

1. Abra o projeto no IDE.

2. Aponte o interpretador Python para o mesmo venv usado para instalar dependências.
3. Execute os scripts (run_comprehensive_analysis → visualize_results → create_pdf_report) no terminal do IDE ou rode como configuração de execução do IDE.
4. Visualize resultados em chattts_analysis/.

---

9 — Exemplo rápido de execução (comandos resumidos)

```
# 1. Preparar
python -m venv .venv
source .venv/bin/activate
pip install -r requirements.txt

# 2. Rodar análise principal
python tools/run_comprehensive_analysis.py

# 3. Gerar gráficos
python tools/visualize_results.py

# 4. Gerar PDF
python tools/create_pdf_report.py
```

Tutorial de Uso – Análise do Repositório ChatTTS com *Flan-T5 Base* e *Twitter-RoBERTa Sentiment*

Este tutorial descreve como replicar um pipeline completo de análise do repositório **ChatTTS**
https://github.com/2noise/ChatTTS
utilizando dois modelos de linguagem especializados:

- **google/flan-t5-base** – Modelo encoder-decoder para interpretação e classificação textual
- **cardiffnlp/twitter-roberta-base-sentiment-latest** – Modelo robusto de análise de sentimentos

---

1. Preparação do Ambiente

1.1. Instalar dependências

pip install transformers torch sentencepiece pandas

Para exibir resultados estruturados:

pip install tabulate

---

2. Clonando o Repositório ChatTTS

O projeto analisado será o **ChatTTS**, mas o pipeline funciona para qualquer repositório.

git clone https://github.com/2noise/ChatTTS

---

3. Modelo 1: google/flan-t5-base

3.1. Objetivo

- Responder perguntas
- Classificar textos em categorias
- Explicar arquivos
- Realizar análises arquiteturais via *prompting*

Este modelo funciona por **instrução** (“Explique…”, “Classifique…”, “Resuma…”).

3.2. Carregando o modelo

```
from transformers import T5Tokenizer, T5ForConditionalGeneration


model_name = "google/flan-t5-base"


tokenizer = T5Tokenizer.from_pretrained(model_name)
model = T5ForConditionalGeneration.from_pretrained(model_name)
```

3.3. Função de inferência

```
def flan_generate(text, instruction):
    prompt = f"{instruction}\n\nTexto:\n{text}"
    inputs = tokenizer(prompt, return_tensors="pt", max_length=512,
truncation=True)
    outputs = model.generate(**inputs, max_new_tokens=200)
    return tokenizer.decode(outputs[0], skip_special_tokens=True)
```

3.4. Exemplos de uso

Analisar arquivo do ChatTTS

```
text = open("ChatTTS/README.md", "r", encoding="utf-8").read()


instruction = "Explique os principais componentes arquiteturais presentes neste
texto."


print(flan_generate(text, instruction))
```

Classificar tipo de arquivo

```
instruction = "Classifique o tipo de arquivo entre: documentação, código,
configuração ou dados."


print(flan_generate(text, instruction))
```

---

4. Modelo 2: cardiffnlp/twitter-roberta-base-sentiment-latest

4.1. Objetivo

- Avaliar sentimento
- Detectar polaridade
- Auxiliar na análise de qualidade de documentação

4.2. Carregar modelo

```
from transformers import AutoTokenizer, AutoModelForSequenceClassification
import torch
import torch.nn.functional as F

model_name = "cardiffnlp/twitter-roberta-base-sentiment-latest"

tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForSequenceClassification.from_pretrained(model_name)
```

4.3. Função de inferência

```
def analyze_sentiment(text):
    inputs = tokenizer(text, return_tensors="pt", truncation=True,
max_length=512)
    outputs = model(**inputs)
    scores = F.softmax(outputs.logits, dim=1)
    labels = ["negative", "neutral", "positive"]
    return {labels[i]: float(scores[0][i]) for i in range(3)}
```

4.4. Exemplo de uso

```
text = open("ChatTTS/README.md", "r", encoding="utf-8").read()
```

print(analyze_sentiment(text))

---

5. Pipeline Completo para Replicação

Passo 1 — Escolher o arquivo

file = "ChatTTS/README.md"

text = open(file, "r", encoding="utf-8").read()

Passo 2 — Executar o FLAN-T5

interpretacao = flan_generate(text, "Explique os conceitos principais do texto")

Passo 3 — Executar o RoBERTa

sentimento = analyze_sentiment(text)

Passo 4 — Consolidar

print("=== INTERPRETAÇÃO ===")

print(interpretacao)

print("\n=== SENTIMENTO ===")

print(sentimento)

---

7. Conclusão

Este tutorial oferece todas as instruções necessárias para replicar, em qualquer IDE, análises utilizando:

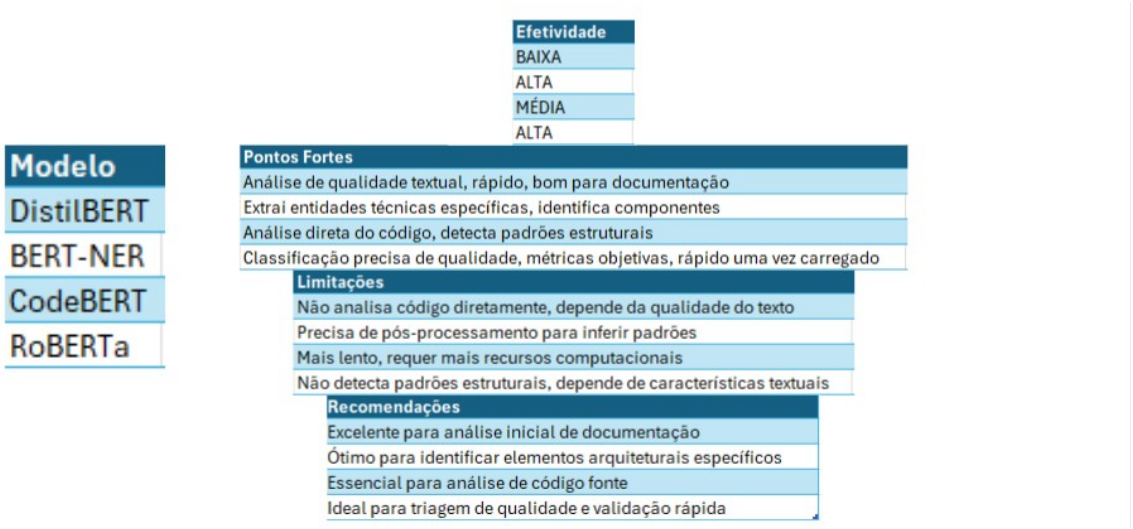
- **RoBERTa** → sentimento, polaridade, análise de qualidade

  FLAN-T5 → interpretação, explicação, classificação

Da mesma forma que o primeiro tutorial analisava o ChatTTS com três modelos, este reproduz a metodologia usando apenas os dois novos modelos. O FLAN-T5 serviu como ferramenta auxiliar para o processamento de texto e analise qualitativa complementar para validação.

O RoBERTa serviu como classificação de qualidade baseado em características textuais do código.

RESULTADOS:



Modelo	Efetividade	Pontos Fortes	Limitações	Recomendações
DistilBERT	BAIXA	Análise de qualidade textual, rápido, bom para documentação	Não analisa código diretamente, depende da qualidade do texto	Excelente para análise inicial de documentação
BERT-NER	ALTA	Extrai entidades técnicas específicas, identifica componentes	Precisa de pós-processamento para inferir padrões	Ótimo para identificar elementos arquiteturais específicos
CodeBERT	MÉDIA	Análise direta do código, detecta padrões estruturais	Mais lento, requer mais recursos computacionais	Essencial para análise de código fonte
RoBERTa	ALTA	Classificação precisa de qualidade, métricas objetivas, rápido uma vez carregado	Não detecta padrões estruturais, depende de características textuais	Ideal para triagem de qualidade e validação rápida

- RoBERTa Sentiment demonstrou alta efetividade na análise de qualidade do código, com pontos fortes que incluem classificação precisa de qualidade, métricas objetivas e processamento rápido após o carregamento inicial. Suas principais limitações são a incapacidade de detectar padrões estruturais e a dependência de características textuais do código. É recomendado como ideal para triagem de qualidade e validação rápida de projetos.

- DistilBERT apresentou baixa efetividade para análise arquitetural, porém mostrou-se excelente para análise de qualidade textual, sendo rápido e eficiente para documentação. Sua principal limitação é não analisar código diretamente, dependendo da qualidade do texto analisado. Recomenda-se seu uso para análise inicial de documentação técnica.

- BERT-NER alcançou alta efetividade na extração de entidades técnicas específicas e identificação de componentes arquiteturais. Como

limitação, requer pós-processamento para inferir padrões a partir das entidades identificadas. É ótimo para identificar elementos arquiteturais específicos em projetos de software.

- CodeBERT demonstrou efetividade média com a vantagem de realizar análise direta do código fonte e detectar padrões estruturais complexos. Suas limitações incluem velocidade de processamento mais lenta e maior demanda por recursos computacionais. É essencial para análise profunda do código fonte.

Observação: FLAN-T5 é um modelo generativo de texto, não um classificador. Diferente dos outros modelos que categorizam ou extraem informações, ele gera respostas textuais a partir de instruções. Na análise arquitetural, atuou como ferramenta auxiliar para processamento de texto e tokenização, mas mostrou limitações para identificar padrões complexos sem prompts muito específicos. É mais versátil, porém menos especializado que os classificadores.