

✓ 1. Installazione pacchetti

```
1 # Execution time: 0s
2 import importlib.util
3 import subprocess
4 def install_if_not_exists(package, version=None):
5     spec = importlib.util.find_spec(package)
6     if spec is None:
7         print(f"Package '{package}' not found. Installing...")
8         if version:
9             subprocess.check_call(['pip', 'install', f"{package}=={version}"])
10        else:
11            subprocess.check_call(['pip', 'install', package])
12        print(f"Package '{package}' installed successfully!")
13    else:
14        print(f"Package '{package}' is already installed.")
```

```
1 # Execution time: 23s - 36s
2 install_if_not_exists('langchain')
```

```
🔄 Package 'langchain' not found. Installing...
Package 'langchain' installed successfully!
```

```
1 # Execution time: 23s - 58s
2 install_if_not_exists('unstructured')
```

```
1 # Execution time: 1m 33s - 2m 33s
2 install_if_not_exists('sentence_transformers')
```

```
1 # Execution time: 37s - 1m 1s
2 install_if_not_exists('chromadb', '0.4.15')
```

```
1 # Execution time: 19s - 27s
2 install_if_not_exists('googletrans', '3.1.0a0') # '4.0.0-rc1'
```

```
1 # Execution time: 15s - 19s
2 install_if_not_exists('pypdf')
```

```
1 # Execution time: TBD
2 install_if_not_exists('nltk')
```

```
1 # Execution time: TBD
2 install_if_not_exists('huggingface_hub')
```

```
1 # Execution time: 8s - 15s
2 install_if_not_exists('tiktoken')
```

```
1 # Execution time: 12s
2 install_if_not_exists('peft')
```

✓ Caratterizzazione gruppo

```
1 class Gruppo:
2     def __init__(self, ID_gruppo, url_git, urls, embeddings_model_name, repo_id, model_dir, n=None, overlap=None, path_to_PDF_files :
3         self.ID_gruppo = ID_gruppo
4         self.url_git = url_git
5         self.urls = urls
6         self.embeddings_model_name = embeddings_model_name
7         self.repo_id = repo_id
8         self.model_dir = model_dir
9         self.n = n
10        self.overlap = overlap
11        self.path_to_PDF_files = path_to_PDF_files
12        self.documents = documents
```

✓ Gruppi

```

1 G01 = Gruppo(
2     ID_gruppo="G01",
3     url_git="https://github.com/LorenzoAlampi/files_for_chatbot",
4     urls=["http://www.cittametropolitana.torino.it/cms/ambiente/qualita-aria/dati-qualita-aria/ipqa",
5           "http://www.comune.torino.it/ambiente/aria/aria_torino/quali-sono-le-cause-dellinquinamento-dellaria.shtml"],
6     embeddings_model_name="all-mpnet-base-v2",
7     repo_id="declare-lab/flan-alpaca-large",
8     model_dir="./flan-alpaca-large/",
9     n=3,
10    overlap=2
11 )

1 G02 = Gruppo( # split_text_into_groups_G02 !!!!!!!
2     ID_gruppo="G02",
3     url_git="https://github.com/s285305/files_for_chatbot.git",
4     urls=["http://www.canidaguardia.com/news/dettaglio.aspx?id=122"],
5     embeddings_model_name="paraphrase-multilingual-mpnet-base-v2", # all-mpnet-base-v2
6     repo_id="google/long-t5-tglobal-base", # declare-lab/flan-alpaca-large
7     model_dir="./long-t5-tglobal-base/" # ./flan-alpaca-large/
8 )

1 G03 = Gruppo(
2     ID_gruppo="G03",
3     url_git="https://github.com/Carmen297259/Dolci_natalizi1.git",
4     urls=["https://www.sfizioso.it/dolci-di-natale-storia/",
5           "https://www.elle.com/it/cucina/g3594/dolci-di-natale-ricette-facili/"
6           ],
7     embeddings_model_name="all-mpnet-base-v2",
8     repo_id="declare-lab/flan-alpaca-base",
9     model_dir="./flan-alpaca-base/",
10    n=3,
11    overlap=2,
12    path_to_PDF_files="Dolci_natalizi1/PDF/"
13 )

1 G04 = Gruppo(
2     ID_gruppo="G04",
3     url_git="https://github.com/CA299283/files_for_chatbot.git",
4     urls=[
5         "https://www.studenti.it/jane-austen-biografia-libri-stile.html",
6         "https://www.skuela.net/letteratura-inglese-1800-1900/austen-biografia.html",
7         "https://www.libroparlato.org/jane-austen/#:~:text=Lo%20stile%20di%20Jane%20Austen&text=Numerosi%20sono%20infatti%20i%20dialogh",
8         "https://www.studenti.it/jane_austen_letteratura_inglese.html#:~:text=Nel%201783%20ha%20iniziato%20a,la%20Abbey%20School%20di%20",
9     ],
10    embeddings_model_name="all-mpnet-base-v2",
11    repo_id="prithivida/parrot_paraphraser_on_T5",
12    model_dir="./parrot_paraphraser_on_T5/",
13    n=4,
14    overlap=3
15 )

1 G05 = Gruppo(
2     ID_gruppo="G05",
3     url_git="https://github.com/MelissaJolanda/files_for_chatbot.git",
4     urls=[
5         "https://www.pianetadesign.it/catalogo/ikea-ufficio-la-nuova-collezione-trotten.php",
6         "https://www.ikea.com/it/it/rooms/home-office/how-to/come-realizzare-la-scrivania-da-gaming-perfetta-per-te-pub147a6c80",
7         "https://www.ikea.com/it/it/cat/uppspel-serie-54984/"],
8     embeddings_model_name="LaBSE",
9     repo_id="declare-lab/flan-alpaca-base",
10    model_dir="./flan-alpaca-base/",
11    n=4,
12    overlap=1
13 )

```

```

1 G08 = Gruppo(
2     ID_gruppo="G08",
3     url_git="https://github.com/federicocolombo12/dante-s-inferno.git",
4     urls=[
5         "https://auralcraive.com/2019/08/13/il-significato-dellinferno-storia-letteratura-religione/",
6         "https://it.wikipedia.org/wiki/Divina_Commedia#Inferno"
7     ],
8     embeddings_model_name="all-mpnet-base-v2",
9     repo_id="declare-lab/flan-alpaca-large",
10    model_dir="./flan-alpaca-large/",
11    n=2,
12    overlap=1,
13    path_to_PDF_files = "dante-s-inferno/pdf/"
14 )

```

```

1 G09 = Gruppo(
2     ID_gruppo="G09",
3     url_git="https://github.com/RebeccaCrini/File_For_Cartoon_ciak.git",
4     urls=[
5         "https://www.animaker.it/blog/10-tipi-di-animazioni/",
6         "https://www.adobe.com/it/creativecloud/animation/discover/computer-animation.html",
7         "https://www.brevestoriadelcinema.org/08-1-le-origini-del-cinema-di-animazione/"
8     ],
9     embeddings_model_name="all-mpnet-base-v2",
10    repo_id="declare-lab/flan-alpaca-base",
11    model_dir="./flan-alpaca-base/",
12    n=3,
13    overlap=2,
14    path_to_PDF_files = "File_For_Cartoon_ciak/PDF/"
15 )

```

```

1 G10 = Gruppo(
2     ID_gruppo="G10",
3     url_git="https://github.com/rblvince/files_for_chatbot.git",
4     urls=[
5         "https://tg24.sky.it/mondo/2023/07/26/olimpiadi-parigi-2024",
6         "https://www.parigi.it/film-ambientati-a-parigi-da-vedere"
7     ],
8     embeddings_model_name="all-mpnet-base-v2",
9     repo_id="declare-lab/flan-alpaca-large",
10    model_dir="./flan-alpaca-large/",
11    n=3,
12    overlap=2,
13 )

```

```

1 G11 = Gruppo(
2     ID_gruppo="G11",
3     url_git="https://github.com/giulia060402/CD_CAPA_bot.git",
4     urls=[
5         "https://it.wikipedia.org/wiki/Caparezza",
6         "https://www.rockit.it/caparezza"
7     ],
8     embeddings_model_name="all-mpnet-base-v2",
9     repo_id="declare-lab/flan-alpaca-large",
10    model_dir="./flan-alpaca-large/",
11    n=3,
12    overlap=2,
13    path_to_PDF_files = "CD_CAPA_bot/PDF/"
14 )

```

```

1 G12 = Gruppo(
2     ID_gruppo="G12",
3     url_git="https://github.com/FraGrill/files_for_chatbot",
4     urls=[
5         "http://www.comune.torino.it/torinogiovani/vivere-a-torino/musei-aperti-a-torino",
6         "https://www.guidatorino.com/musei-torino-elenco-completo/"
7     ],
8     embeddings_model_name="BAAI/bge-base-en-v1.5",
9     repo_id="declare-lab/flan-alpaca-large",
10    model_dir="./flan-alpaca-large/",
11    n=3,
12    overlap=2,
13 )

```

```

1 G13 = Gruppo(
2     ID_gruppo="G13",
3     url_git="https://github.com/ncstaran/files_for_chatbot.git",
4     urls=[
5         "https://didattica.polito.it/pls/portal30/gap.pkg_guide.viewGap?p_cod_ins=04AFQPC&p_a_acc=2024&p_header=S&p_lang=IT&multi=N",
6         "https://elite.polito.it/teaching/04afqpc-bdcin/intro",
7     ],
8     embeddings_model_name="paraphrase-multilingual-mpnet-base-v2",
9     repo_id="declare-lab/flan-alpaca-base",
10    model_dir="./flan-alpaca-base/",
11    n=3,
12    overlap=2,
13 )

1 G14 = Gruppo(
2     ID_gruppo="G14",
3     url_git="https://github.com/carlottazzz/files_for_chatbot.git",
4     urls=[
5         "https://www.dovesciare.it/dove-sciare-piemonte",
6         "https://serendipitsite.com/2021/07/13/piemonte-valli-piemontesi-quali-scegliere-montagna/",
7         "https://it.wikiloc.com/percorsi/parapendio/italia/piemonte"
8     ],
9     embeddings_model_name="multi-qa-mpnet-base-dot-v1",
10    repo_id="declare-lab/flan-alpaca-large",
11    model_dir="./flan-alpaca-large/",
12    n=5,
13    overlap=2,
14 )

1 G15 = Gruppo(
2     ID_gruppo="G15",
3     url_git="https://github.com/StruggledMick/GuidaMichelinTorino.git",
4     urls=[
5         "https://www.dissapore.com/ristoranti/ristoranti-stellati-di-torino-2023-cosa-si-mangia-e-quanto-costano-i-premiati-dalla-guida",
6         "https://www.torinofree.it/enogastronomia/ristoranti-una-stella-michelin-torino-2023.html",
7     ],
8     embeddings_model_name="paraphrase-multilingual-mpnet-base-v2",
9     repo_id="declare-lab/flan-alpaca-large",
10    model_dir="./flan-alpaca-large/",
11    n=4,
12    overlap=3,
13    path_to_PDF_files = "GuidaMichelinTorino/PDF/"
14 )

1 G16 = Gruppo(
2     ID_gruppo="G16",
3     url_git="https://github.com/AntoLindo/files_for_chatbot.git",
4     urls=[
5         "https://www.lapasticcionavegana.it/ricette-vegan/category/Salate",
6         "https://www.cookist.it/ricette-vegane/"
7     ],
8     embeddings_model_name="all-MiniLM-L6-v2",
9     repo_id="declare-lab/flan-alpaca-large",
10    model_dir="./flan-alpaca-large/",
11    n=3,
12    overlap=2,
13 )

1 G17 = Gruppo(
2     ID_gruppo="G17",
3     url_git="https://github.com/jacopolucci/files_for_chatbot.git",
4     urls=[
5         "https://www.informazioneambiente.it/squalo/",
6         "https://squali.jimdofree.com/anatomia-anatomy/",
7         "https://imieianimali.it/il-comportamento-dello-squalo/"
8     ],
9     embeddings_model_name="all-mpnet-base-v2",
10    repo_id="declare-lab/flan-alpaca-large",
11    model_dir="./flan-alpaca-large/",
12    n=3,
13    overlap=2,
14 )

```

```

1 G19 = Gruppo(
2     ID_gruppo="G19",
3     url_git="https://github.com/NataleFlavia/files_for_chatbot.git",
4     urls=[
5         "https://it.wikipedia.org/wiki/RuPaul%27s_Drag_Race"
6     ],
7     embeddings_model_name="bert-base-nli-mean-tokens",
8     repo_id="declare-lab/flan-alpaca-base",
9     model_dir="./flan-alpaca-base",
10    n=3,
11    overlap=2,
12 )

1 G20 = Gruppo(
2     ID_gruppo="G20",
3     url_git="https://github.com/botBDS/files_for_chatbot.git",
4     urls=[
5         "https://it.wikipedia.org/wiki/Centro_sperimentale_di_cinematografia",
6         "https://cinecittanews.it/il-futuro-del-csc-nuove-sedi-sul-territorio-piattaforma-e-learning-e-spirito-green/",
7         "https://it.wikipedia.org/wiki/Cineteca_Nazionale",
8         "https://www.hollywoodreporter.com/movies/movie-news/top-global-film-schools-2023-1235559015/",
9         "https://cineuropa.org/en/schoolprofile/200285/"
10    ],
11    embeddings_model_name="stsb-xml-r-multilingual",
12    repo_id="declare-lab/flan-alpaca-large",
13    model_dir="./flan-alpaca-large/",
14    n=3,
15    overlap=2,
16 )

1 G21 = Gruppo(
2     ID_gruppo="G21",
3     url_git="https://github.com/lorenzoepi/files_for_chatbot.git",
4     urls=[
5         "https://farmaciagaudiana.it/allergia-ai-gatti-e-al-loro-pelo-come-convivere",
6         "https://www.docpeter.it/blog/post/allergie-ai-gatti-cause-sintomi-e-rimedi",
7         "https://zampettaverde.it/blog/razze-gatti-ipoallergenici/",
8         "https://oggiscienza.it/2021/09/17/allergie-cani-gatti/index.html"
9     ],
10    embeddings_model_name="all-mpnet-base-v2",
11    repo_id="declare-lab/flan-alpaca-large",
12    model_dir="./flan-alpaca-large/",
13    n=2,
14    overlap=1,
15 )

1 G22 = Gruppo(
2     ID_gruppo="G22",
3     url_git="https://github.com/PerAdotz/files_for_chatbot.git",
4     urls=[
5         "https://camminiditalia.org/cammini-italia/",
6         "https://www.komoot.com/guide/537/hiking-in-italy",
7         "https://www.trekking.it/news/line-latitante-dei-cammini/"
8     ],
9     embeddings_model_name="bert-base-nli-mean-tokens",
10    repo_id="declare-lab/flan-alpaca-base",
11    model_dir="./flan-alpaca-base",
12    n=3,
13    overlap=2,
14 )

1 G23 = Gruppo( # split_text_into_groups_G23 !!!!!!!
2     ID_gruppo="G23",
3     url_git="https://github.com/T0mB0t/files_for_chatbot.git",
4     urls=[
5         "https://it.wikipedia.org/wiki/American_Psycho_(film)",
6         "https://cinemaserietv.it/film/american-psycho-la-spiegazione-del-finale-del-film-con-christian-bale/",
7         "https://it.wikipedia.org/wiki/Mary_Harron",
8         "https://it.wikipedia.org/wiki/Christian_Bale"
9     ],
10    embeddings_model_name="all-MiniLM-L6-v2", # all-mpnet-base-v2
11    repo_id="google/flan-t5-base", # declare-lab/flan-alpaca-large
12    model_dir="./flan-t5-base", # ./flan-alpaca-large/
13 )

```

```

1 G24 = Gruppo(
2     ID_gruppo="G24",
3     url_git="https://github.com/LorZampy/files_for_chatbot.git",
4     urls=[
5         "https://www.boxofficemojo.com/weekly/by-year/2019/",
6         "https://www.boxofficemojo.com/year/2019/",
7         "https://screenrant.com/most-anticipated-movies-2019/"
8     ],
9     embeddings_model_name="all-mpnet-base-v2",
10    repo_id="declare-lab/flan-alpaca-large",
11    model_dir="./flan-alpaca-large/",
12    n=3,
13    overlap=2,
14 )

1 G25 = Gruppo(
2     ID_gruppo="G25",
3     url_git="https://github.com/CeciliaRinaudo/files_for_chatbot.git",
4     urls=[
5         "https://www.elle.com/it/magazine/arte/a40241528/banksy-opere/",
6         "https://leonardo.it/curiosita/chi-e-banksy/",
7         "https://www.exibart.com/mercato/banksy-mania-le-opere-piu-costose-dell-artista-di-bristol/",
8         "https://glamcasamagazine.it/murales-di-banksy-i-piu-famosi-ecco-dove-trovarli/",
9         "https://seven7art.it/?p=2130",
10    ],
11    embeddings_model_name="all-mpnet-base-v2",
12    repo_id="MBZUI/LaMini-Flan-T5-783M", # declare-lab/flan-alpaca-large
13    model_dir="./LaMini-Flan-T5-783M/", # ./flan-alpaca-large/
14    n=3,
15    overlap=2,
16 )

1 G27 = Gruppo(
2     ID_gruppo="G27",
3     url_git="https://github.com/ElisaRuggio/files_for_chatbot.git",
4     urls=[
5         "https://www.mymovies.it/cinema/",
6     ],
7     embeddings_model_name="multi-qa-mpnet-base-cos-v1",
8     repo_id="declare-lab/flan-alpaca-base",
9     model_dir="./flan-alpaca-base/",
10    n=3,
11    overlap=2,
12 )

1 G28 = Gruppo(
2     ID_gruppo="G28",
3     url_git="https://github.com/tbboom01/file_for_chatbot",
4     urls=[
5         "https://lol.fandom.com/wiki/New_To_League/Welcome",
6         "https://en.wikipedia.org/wiki/Swiss-system_tournament"
7     ],
8     embeddings_model_name="paraphrase-multilingual-mpnet-base-v2",
9     repo_id="declare-lab/flan-alpaca-large",
10    model_dir="./flan-alpaca-large/",
11    n=3,
12    overlap=2,
13    path_to_PDF_files = "file_for_chatbot/PDF/"
14 )

1 G30 = Gruppo(
2     ID_gruppo="G30",
3     url_git="https://github.com/twmiaz/file.git",
4     urls=[
5         "https://collebianco.it/alimentazione/storia-della-mozzarella-di-bufala/",
6         "https://www.foodinfo.it/food/mozzarella-di-bufala-storia-e-origini-delloro-bianco-della-campania/#:~:text=I%20primi%20documenti",
7         "https://it.wikipedia.org/wiki/Mozzarella_di_bufala_campana",
8         "https://www.cookist.it/quanto-deve-costare-una-mozzarella-e-i-trucchetti-per-sceglierla-secondo-un-vero-esperto/#:~:text=Qual",
9         "https://www.saporie.com/ricetta-base/preparare-i-classici/come-si-mangia-la-mozzarella-di-bufala"
10    ],
11    embeddings_model_name="paraphrase-multilingual-mpnet-base-v2",
12    repo_id="declare-lab/flan-alpaca-large",
13    model_dir="./flan-alpaca-large/",
14    n=3,
15    overlap=2,
16    path_to_PDF_files = "file/PDF/"
17 )

```

✓ Variabile

```
1 gruppo_corrente = G25
2 print(f"Stai testando il gruppo {gruppo_corrente.ID_gruppo}:")
3 for attr, val in vars(gruppo_corrente).items():
4     print(f"{attr}: {val}")
```

✓ Caricamento dei documenti

```
1 # Execution time: 22s - 59s
2 # N.B. Prima di eseguire per la prima volta riavviare la sessione!
3 !rm -rf files_for_chatbot
4 !git clone $gruppo_corrente.url_git
5 from langchain.document_loaders import UnstructuredURLLoader
6 from langchain.document_loaders import PyPDFLoader
7 import os
8
9 loader = UnstructuredURLLoader(urls=gruppo_corrente.urls)
10 gruppo_corrente.documents = loader.load()
11 documenti_sorgente = []
12 for url in gruppo_corrente.urls:
13     documenti_sorgente.append(url)
14
15 files = os.listdir(gruppo_corrente.path_to_PDF_files)
16
17 for f in files:
18     # if f != 'ricette-vegane.pdf':
19         documenti_sorgente.append(f)
20         loader = PyPDFLoader(gruppo_corrente.path_to_PDF_files + f)
21         gruppo_corrente.documents += loader.load_and_split()
22
23 print('-----')
24 print('Documenti sorgente:')
25 i = 0
26 for doc in documenti_sorgente:
27     i += 1
28     print(f"{i} {doc}")
29 print('-----')
30
31 print(f"Sono stati caricati {len(gruppo_corrente.documents)} documenti.")
32 print('-----')
33
34 i = 0
35 for doc in gruppo_corrente.documents:
36     print(f"{i} Metadati:", doc.metadata)
37     i += 1
38     first_words = " ".join(doc.page_content.split()[:50])
39     print(first_words, '...')
40     print('-----')
```

✓ Suddivisione dei documenti

✓ Definizione split_text_into_groups

```

1 # Execution time: 0s
2 import nltk
3 nltk.download('punkt')
4
5 # default
6 def split_text_into_groups(text, n=gruppo_corrente.n, overlap=gruppo_corrente.overlap, lang="italian"):
7     sentences = nltk.sent_tokenize(text, language=lang)
8     groups = []
9     for i in range(0, len(sentences), n - overlap):
10         if i + n < len(sentences) or len(groups) == 0:
11             group = ' '.join(sentences[i:i + n])
12             groups.append(group)
13     return groups
14
15 # G02
16 def split_text_into_groups_G02(text, n=300, overlap=50, lang="italian"):
17     groups = []
18     for i in range(0, len(text), n - overlap):
19         group = text[i:i + n]
20         groups.append(group)
21     return groups
22
23 # G23
24 def split_text_into_groups_G23(text, max_len=512, overlap=0):
25     char_groups = []
26     for i in range(0, len(text), max_len - overlap):
27         if i + max_len < len(text) or len(groups) == 0:
28             char_group = text[i:i + max_len]
29             char_groups.append(char_group)
30     return char_groups

```

✓ Suddivisione

```

1 # Execution time: 10s
2 import matplotlib.pyplot as plt
3 from langchain.docstore.document import Document
4
5 doc_id = 0
6 documentsSplitted = list()
7 i = 0
8 x = []
9 y = []
10 for original_doc in gruppo_corrente.documents:
11     doc_chunk_list = split_text_into_groups(original_doc.page_content) # dipende dal gruppo
12     print(f"Il documento {doc_id} è stato suddiviso in {len(doc_chunk_list)} chunk.")
13
14     for group_content in doc_chunk_list:
15         chunk_metadata = original_doc.metadata
16         chunk_metadata['id'] = 0
17         docSplitted = Document(page_content=group_content, metadata=chunk_metadata)
18         documentsSplitted.append(docSplitted)
19         x.append(i)
20         n_parole = len(group_content.split())
21         y.append(n_parole)
22         i += 1
23
24     doc_id += 1
25
26 print()
27 print(f"In totale ci sono {len(documentsSplitted)} chunk.")
28
29 if len(documentsSplitted) < 500:
30     plt.bar(x, y)
31     plt.show()

```

✓ Traduzione dei chunk

✓ Definizione google_translator


```

1 # Execution time: 2s
2
3 from googletrans import Translator
4
5 def google_translator(text, from_code="it", to_code="en"):
6     translator = Translator()
7     translation = translator.translate(text, dest=to_code, src=from_code).text
8     return translation

```

✓ Traduzione

```

1 # Execution time: 2m 26s - 17m
2
3 import pandas as pd
4 import ipywidgets as widgets
5 from IPython.display import display
6 from tqdm import tqdm
7
8 output_widget = widgets.Output()
9 display(output_widget)
10
11 documentsSplitted_en = list()
12 documentsSplitted_en_ita_dict = dict()
13
14 i = 0
15 diff = 0
16 perc_avanzamento = 0
17 for doc in tqdm(documentsSplitted[i:], desc='Progresso'):
18     i += 1
19     translated_text = google_translator(' '.join(doc.page_content.split()))
20     documentsSplitted_en_ita_dict[translated_text] = doc.page_content
21     docSplitted_en = Document(page_content=translated_text, metadata=doc.metadata)
22     documentsSplitted_en.append(docSplitted_en)
23     diff_new = len(documentsSplitted_en) - len(documentsSplitted_en_ita_dict)
24     if diff_new != diff:
25         print(f"il chunk {i-1} non è stato aggiunto a documentsSplitted_en_ita_dict.")
26         diff = diff_new
27         with output_widget:
28             output_widget.clear_output(wait=True)
29             perc_avanzamento = round(i/len(documentsSplitted)*100, 2)
30             # print(f"Iterazione: {i}/{len(documentsSplitted)} ({perc_avanzamento}%")
31
32     pass
33
34 df = pd.DataFrame(documentsSplitted_en_ita_dict.items(), columns=['Chiave', 'Valore'])
35 print("Contenuto del dizionario documentsSplitted_en_ita_dict:")
36 display(df)

```

✓ Definizione dei modelli

```

1 # Execution time: 7 - 9s
2 from transformers import AutoTokenizer
3 from sentence_transformers import SentenceTransformer, util
4 from langchain.embeddings import HuggingFaceEmbeddings
5 from langchain.vectorstores import Chroma
6 from huggingface_hub import snapshot_download
7 from langchain.llms import HuggingFacePipeline
8 from langchain.chains import RetrievalQA
9 import warnings
10
11 warnings.filterwarnings("ignore", category=UserWarning)

```

✓ Controllo token

```

1 # Execution time: 0s
2
3 def get_model_max_len(model_name=gruppo_corrente.embeddings_model_name):
4     model = SentenceTransformer('sentence-transformers/' + model_name)
5     tokenizer = model.tokenizer
6     max_len = tokenizer.model_max_length
7     return max_len

```

```

1 # Execution time: 0s
2
3 def count_text_token(text, model_name=gruppo_corrente.embeddings_model_name):
4     tokenizer = AutoTokenizer.from_pretrained('sentence-transformers/' + model_name)
5     tokens = tokenizer(text)
6     num_tokens = len(tokens['input_ids'])
7     return num_tokens

1 # Execution time: 27s
2
3 import ipywidgets as widgets
4 from IPython.display import display
5
6 # Widget per mostrare l'output
7 output_widget = widgets.Output()
8 display(output_widget)
9
10 # Ottieni la lunghezza massima consentita dal modello di embedding
11 max_len_model = get_model_max_len()
12
13 i = 0
14 for doc in documents splitted_en:
15     i += 1
16     if count_text_token(doc.page_content) > max_len_model:
17         print(f"Il chunk {i} supera la lunghezza massima del modello di embedding")
18
19     with output_widget:
20         output_widget.clear_output(wait=True)
21         print(f"Iterazione: {i}/{len(documents splitted_en)}")

```

✓ Creazione del retriever

```

1 # Execution time: 2s - 32s
2 print("Inizio la creazione dell'embedding model.")
3
4 embedding_model = HuggingFaceEmbeddings(model_name=gruppo_corrente.embeddings_model_name,
5                                         model_kwargs={"device": "cpu"},
6                                         encode_kwargs={"normalize_embeddings": True})
7
8 print("L'embedding model è stato creato correttamente.")

1 # Execution time: 28s - 17m
2 print('Inizio la creazione del vector store.')
3
4 db = Chroma.from_documents(documents splitted_en,
5                             embedding_model,
6                             collection_metadata={"hnsw:space": "cosine"})
7
8 print('Il vector store è stato creato correttamente.')

1 # Execution time: 32s - 3m 31s
2 if os.path.exists(gruppo_corrente.model_dir):
3     print('Il LLM è già stato scaricato.')
4 else:
5     print('Inizio a scaricare il LLM.')
6     !mkdir $gruppo_corrente.model_dir
7     snapshot_download(repo_id=gruppo_corrente.repo_id,
8                       local_dir=gruppo_corrente.model_dir,
9                       local_dir_use_symlinks=False,
10                      token="hf_...")
11 print('Il LLM è stato scaricato correttamente.')

1 # Execution time: 8s - 25s
2 print('Inizio la creazione del LLM.')
3
4 llm = HuggingFacePipeline.from_model_id(model_id=gruppo_corrente.model_dir,
5                                         task='text2text-generation',
6                                         model_kwargs={"temperature": 0.60,
7                                                         "min_length": 35,
7                                                         "max_length": 500,
7                                                         "repetition_penalty": 5.0})
8
9
10
11 print('Il LLM è stato creato correttamente.')

```

```

1 # Execution time: 0s
2
3 qa = RetrievalQA.from_chain_type(llm=llm,
4                                 chain_type="stuff",
5                                 retriever=db.as_retriever(),
6                                 return_source_documents=True,
7                                 verbose=False)

```

✓ Produzione del risultato

✓ Estrazione domande e risposte originali

```

1 # Execution time: 0s
2
3 import pandas as pd
4 # questions_df = pd.read_excel("/content/TSI_AR_ER_G02_Domande risposte.xlsx")
5 try:
6     questions_df = pd.read_csv("files_for_chatbot/Domande chatbot.csv")
7 except FileNotFoundError:
8     # Se il file "Domande chatbot.csv" non è trovato, prova con "Domande e risposte.csv"
9     try:
10         questions_df = pd.read_csv("files_for_chatbot/Domande e risposte.csv")
11     except FileNotFoundError:
12         print("Il file 'Domande e risposte.csv' non è stato trovato.")
13         try:
14             nome_file_domande = input("Inserisci il path del file:")
15             print(nome_file_domande)
16             questions_df = pd.read_csv(nome_file_domande)
17         except FileNotFoundError:
18             print(f"Il file {nome_file_domande} non è stato trovato.")
19 domande = list(questions_df['Domande'])
20 domande_risposte_dict = questions_df.to_dict(orient='records')
21 print('-----')
22 i = 0
23 for domanda in domande:
24     i += 1
25     print(f"{i}) {domanda}")
26 print(domande_risposte_dict)

```

✓ Soluzione di riserva per le domande

```

1 # in alternativa (caricamento domande manuale)
2 domande = [
3     "Com'è chiamato il tipo di cane più efficace per fare la guardia?",
4     "Nominare 3 nomi di cani da guardiania",
5     "Qual è il cane da caccia con maggiori attitudini alla guardia?",
6     "Per fare la guardia dentro casa quale tipo di cane è più indicato?",
7     "Quale cane è un ottimo avvisatore?",
8     "Un cane da guardia è in ogni caso pericoloso se si hanno bambini in casa?",
9     "Quali sono le condizioni per una convivenza uomo-cane pacifica e soddisfacente?",
10    "Fino a quanto i cuccioli dovrebbero rimanere con la madre?",
11    "Al rientro in casa chi dovrebbe avere la priorità di saluto per una corretta educazione del cane?",
12    "La capacità di fare la guardia può essere appresa da qualsiasi cane?"
13 ]

```

✓ Definizione get_chat_response

```

1 # Execution time: 0s
2
3 # Funzione per ottenere una risposta dalla chat
4 def get_chat_response(text, qa=qa, lang='it'):
5     with warnings.catch_warnings():
6         warnings.filterwarnings("ignore")
7         if lang == "it":
8             text_en = google_translator(text)
9             res = {}
10            res = qa(text_en)
11            res["query"] = text
12            res["result"] = google_translator(res["result"], from_code="en", to_code="it")
13            source_documents_translated = list()
14            for d in res['source_documents']:
15                d.page_content = documentsSplitted_en_ita_dict[d.page_content]
16                source_documents_translated.append(d)
17            res["source_documents"] = source_documents_translated
18        else:
19            res = qa(text)
20    return res

```

Calcoli

```

1 # Execution time: 1m 30s - 6m 1s
2
3 import ipywidgets as widgets
4 from IPython.display import display
5
6 # Widget per mostrare l'output
7 output_widget = widgets.Output()
8 display(output_widget)
9
10 res_list = []
11 i = 0
12 for domanda in domande[:10]:
13     i += 1
14
15     risposta = get_chat_response(domanda)
16     print('Risposta ottenuta per -->', domanda)
17
18     res_list.append(risposta)
19
20     with output_widget:
21         output_widget.clear_output(wait=True)
22         print(f"Iterazione: {i}/10")

```

Formattazione risultati

```

1 # Execution time: 0s
2 import pandas as pd
3 df_test = pd.DataFrame(columns=['ID gruppo', 'ID domanda', 'ID tupla', 'Domanda', 'Risposta originale', 'Risposta del chatbot', 'Chur
4
5 D = 1
6 for res in res_list[:10]:
7     ID_domanda = gruppo_corrente.ID_gruppo + "_D" + str(D)
8     domanda = res['query']
9
10    for riga in domande_risposte_dict:
11        if riga['Domande'] == domanda: # dipende dal gruppo
12            risposta_originale = riga['Risposte'] # dipende dal gruppo
13            break
14
15    risposta = res['result']
16    chunk_set = res['source_documents']
17
18    C = 1
19    for chunk in chunk_set:
20        ID_tupla = gruppo_corrente.ID_gruppo + "_D" + str(D) + "_C" + str(C)
21        nuova_riga = {'ID gruppo': gruppo_corrente.ID_gruppo,
22                      'ID domanda': ID_domanda,
23                      'ID tupla': ID_tupla,
24                      'Domanda': domanda,
25                      'Risposta originale': risposta_originale,
26                      'Risposta del chatbot': risposta,
27                      'Chunk': chunk.page_content,
28                      'Documento': chunk.metadata['source'],
29                      'Valutazione': 'TBD',
30                      'Note': ''}
31        df_test = pd.concat([df_test, pd.DataFrame([nuova_riga])], ignore_index=True)
32        C += 1
33
34    D += 1

1 nome_file = gruppo_corrente.ID_gruppo + ".csv"
2 print(nome_file)
3 df_test.to_csv(nome_file, index=False, escapechar='\\')
4 file_path = '/content/' + nome_file
5 pd.read_csv(file_path)

```

▼ Test

▼ Stampa attributi gruppo_corrente

```

1 for attr, val in vars(gruppo_corrente).items():
2     print(f"{attr}: {val}")
3 print('-----')

```

▼ Caricamento singolo documento

```

1 documents_test = []
2 loader1 = PyPDFLoader('/content/The 50 Most Anticipated Movies of 2019.pdf')
3 #loader2 = PyPDFLoader('/content/boxofficeUSA_2019_flopsHits (2).pdf')
4 documents_test += loader1.load_and_split()
5 #documents_test += loader2.load_and_split()
6
7 print(len(documents_test))
8 for doc in documents_test:
9     print(doc.metadata)
10    print('-----')

```

```

1 # Execution time: 22s - 59s
2 # N.B. Prima di eseguire per la prima volta riavviare la sessione!
3 from langchain.document_loaders import UnstructuredURLLoader
4 from langchain.document_loaders import PyPDFLoader
5 import os
6
7 # files_test = ['/content/The 50 Most Anticipated Movies of 2019.pdf', '/content/boxofficeUSA_2019_flopsHIIts (2).pdf']
8 # files_test = ['/content/files_for_chatbot/PDF/GlobalBoxOfficeUSA_19.pdf']
9 # files_test = ['/content/The 50 Most Anticipated Movies of 2019.pdf']
10 files_test = ['/content/boxofficeUSA_2019_flopsHIIts (2).pdf']
11 documents_test = []
12 documenti_sorgente_test = []
13 for f in files_test:
14     if f != 'ricette-vegane.pdf':
15         documenti_sorgente_test.append(f)
16         loader_test = PyPDFLoader(f)
17         documents_test += loader_test.load_and_split()
18
19 print('-----')
20 print('Documenti sorgente:')
21 i = 0
22 for doc in documenti_sorgente_test:
23     i += 1
24     print(f"{i}) {doc}")
25 print('-----')
26
27 print(f"Sono stati caricati {len(documents_test)} documenti.")
28 print('-----')
29
30 i = 0
31 for doc in documents_test:
32     print(f"{i}) Metadati:", doc.metadata)
33     i += 1
34     first_words = " ".join(doc.page_content.split()[:50])
35     print(first_words, '...')
36     print('-----')

```

✓ Visualizzazione singolo documento

```

1 # Chiedi all'utente di inserire l'indice desiderato
2 doc_id = int(input("Inserisci l'indice del documento da visualizzare: "))
3 print('-----')
4
5 # Verifica se l'indice è valido
6 if 0 <= doc_id < len(gruppo_corrente.documents):
7     # Accedi all'elemento con l'indice specificato e stampalo
8     documento = gruppo_corrente.documents[doc_id]
9     # print("Il documento corrispondente all'indice", doc_id, "è:")
10    # print("Id doc: ", documento.metadata['id'])
11    print(documento.page_content)
12 else:
13    print("L'indice inserito non è valido.")

```

✓ Visualizzazione singolo chunk

```

1 # Chiedi all'utente di inserire l'indice desiderato
2 chunk_id = int(input("Inserisci l'indice del chunk da visualizzare: "))
3 print('-----')
4
5 # Verifica se l'indice è valido
6 if 0 <= chunk_id < len(documents_split):
7     # Accedi all'elemento con l'indice specificato e stampalo
8     chunk = documents_split[chunk_id]
9     # print("Il documento corrispondente all'indice", doc_id, "è:")
10    # print(chunk.metadata['id'])
11    print(chunk.page_content)
12 else:
13    print("L'indice inserito non è valido.")

```

✓ Visualizzazione tutti chunk

```

1 i = 0
2 for chunk in documentsSplitted_en:
3     print(f"{i}) {' '.join(chunk.page_content.split())}")
4     if chunk.page_content is None:
5         print('None')
6         break
7     print('-----')
8     i += 1

```

Traduzione

```

1 def google_translator_test(text, from_code="it", to_code="en"):
2     if text is None:
3         return None # Restituisci None se il testo in input è None
4     translator = Translator()
5     translation = translator.translate(text, dest=to_code, src=from_code).text
6     return translation
7
8
9
10 # Execution time: 2m 26s - 11m
11
12 import pandas as pd
13 import ipywidgets as widgets
14 from IPython.display import display
15
16 output_widget = widgets.Output()
17 display(output_widget)
18
19 documentsSplitted_en = list()
20 documentsSplitted_en_ita_dict = dict()
21
22 i = 170
23 for doc in documentsSplitted[i:]:
24     i += 1
25     # print(f"{i}) {' '.join(doc.page_content.split()[:50])}")
26     if isinstance(doc.page_content, str):
27         # translated_text = google_translator(doc.page_content)
28         translated_text = google_translator(' '.join(doc.page_content.split()))
29         print(i)
30     else:
31         # Gestisci il caso in cui doc.page_content non sia una stringa
32         print('***** ERRORE *****')
33     # translated_text = google_translator_test(doc.page_content)
34     # print(f"{i}) {' '.join(translated_text.split()[:50])}")
35     documentsSplitted_en_ita_dict[translated_text] = doc.page_content
36     docSplitted_en = Document(page_content=translated_text, metadata=doc.metadata)
37     documentsSplitted_en.append(docSplitted_en)
38
39
40 translated_text = google_translator(' '.join(documentsSplitted[53].page_content.split()))
41 # translated_text = google_translator(documentsSplitted[53].page_content)
42 print(translated_text)

```

```
1 import pandas as pd
2 import ipywidgets as widgets
3 from IPython.display import display
4
```