La projection en 2D via t-SNE montre que les documents sont regroupés en 5 clusters bien distincts, chacun correspondant à un topic appris automatiquement par le modèle LDA. Cela valide la cohérence des topics détectés. Chaque point est une question Stack Overflow, et sa couleur indique le topic dominant identifié.

API :

Étape 1 : Créer la structure de ton projet

1. Créer les dossiers et fichiers de base (photo)
2. Contenu du fichier README.md de départ
3. Fichier requirements.txt initial

Étape 2 : Créer l'API avec FastAPI (point d'entrée main.py)

1. Installer FastAPI et Uvicorn
2. Contenu initial du fichier app/main.py
3. Contenu initial du fichier app/model.py
4. Lancer le serveur localement (http://127.0.0.1:8000/docs#/default/predict\_keywords\_predict\_post)

Fichier test\_main :

Quelques points rapides qui confirment que ça marche bien :

* Tu utilises TestClient de FastAPI pour simuler des requêtes HTTP, c’est nickel.
* Tu patch la fonction predict\_tags avec unittest.mock.patch pour isoler le test et ne pas dépendre de la vraie fonction ML.
* Tu testes la réponse HTTP (status code 200) et le contenu JSON, ça couvre bien l’essentiel.
* Tu as un test avec un texte valide et un test avec un texte vide, c’est un bon début pour couvrir des cas courants.

Fichier python-app :

**Ce que fait ce fichier :**

* Il se déclenche à chaque push ou pull\_request sur la branche main
* Il installe Python 3.11
* Il installe les dépendances (pip install -r requirements.txt)
* Il lance tes tests (pytest tests/)

Rôle :

**📄 README.md**

**But :** C’est la page d’accueil de ton projet sur GitHub.

**Contenu habituel :**

* Objectif du projet.
* Instructions pour installer les dépendances.
* Exemple d'utilisation de l’API.
* Explication de la structure des dossiers.
* Comment lancer l'API et les tests.

🧠 **Important pour** : les évaluateurs, les collègues, les utilisateurs.

**📦 requirements.txt**

**But :** Liste tous les **packages Python** nécessaires pour faire fonctionner ton projet.

**Exemples typiques de contenu :**

fastapi

uvicorn

mlflow

scikit-learn

pandas

httpx

📌 Utilisé automatiquement par pip install -r requirements.txt

**🚀 main.py (dans app/)**

**But :** C’est le **point d’entrée de l’API** avec FastAPI.

**Contenu habituel :**

* Création de l’objet FastAPI.
* Définition des routes (endpoints), par exemple :
* @app.post("/predict")
* def predict(data: InputData):
* ...

⚙️ C’est ce fichier qui est lancé pour démarrer l’API (souvent avec uvicorn app.main:app).

**🧠 model.py (dans app/)**

**But :** Contient **la logique de prédiction**.

**Ce qu’on y trouve généralement :**

* Chargement d’un modèle ML enregistré avec MLflow.
* Fonction predict\_tags(text) qui applique le modèle à un texte.

🔁 **Séparer** la logique métier du code de l’API est une bonne pratique.

**🧪 test\_main.py (dans tests/)**

**But :** Contient les **tests unitaires** de l’API.

**Utilisé par :** pytest (en local ou via GitHub Actions)

**Exemple de test :**

from fastapi.testclient import TestClient

from app.main import app

client = TestClient(app)

def test\_predict():

response = client.post("/predict", json={"text": "Hello world"})

assert response.status\_code == 200

assert "tags" in response.json()

✅ Garantit que l’API fonctionne toujours après des modifications.

**🔁 .github/workflows/python-app.yml**

**But :** Fichier de **configuration de GitHub Actions** (CI/CD).

**Fonctionnement :**

* À chaque **push ou pull request** sur master, GitHub :
  1. Installe Python.
  2. Installe les dépendances.
  3. Exécute les tests.

**Extrait typique :**

on:

push:

branches: [ master ]

pull\_request:

branches: [ master ]

🤖 Automatise les tests → rassure sur la stabilité du projet.

**📁 Structure typique du projet**

PROJET5/

├── app/

│ ├── main.py <- API FastAPI

│ └── model.py <- Logique de prédiction

├── tests/

│ └── test\_main.py <- Tests unitaires de l’API

├── requirements.txt <- Dépendances du projet

├── README.md <- Description du projet

└── .github/

└── workflows/

└── python-app.yml <- GitHub Actions

**Synthèse des approches supervisées**

| **Approche** | **Description** | **F1-score (micro)** | **F1-score (samples)** | **Observations** |
| --- | --- | --- | --- | --- |
| **TF-IDF + LogReg** | Bag-of-Words + OneVsRest(LogReg) | **0.5755** | **0.5043** | Baseline robuste, rapide |
| **Word2Vec + LogReg** | Moyenne vecteurs Word2Vec + LogReg | 0.2733 | 0.2592 | Encode peu la sémantique globale |
| **BERT + LogReg** | Embedding BERT (MiniLM-L6-v2) | **0.6095** | **0.5968** | Meilleur modèle à ce jour |
| **USE (TensorFlow)** | Universal Sentence Encoder | ❌ | ❌ | Incompatible avec Python 3.13 / Windows |

Test\_main :

**Explications rapides**

* On teste chaque endpoint avec un texte valide
* On teste que la réponse est bien un JSON avec la clé keywords et que c’est une liste
* On teste la gestion des textes vides (doit retourner une liste vide)
* On teste la validation Pydantic (si le champ text est absent, ça renvoie 422)
* On teste aussi la gestion d’un JSON invalide